Core 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

Tables

Introduction

Framework	/System/Library/Frameworks/CoreServices
Header file directories	/System/Library/Frameworks/CoreServices.framework/Headers
Declared in	Aliases.h BackupCore.h CFFTPStream.h CFHTTPAuthentication.h CFHTTPMessage.h CFHost.h CFNetDiagnostics.h CFNetServices.h CFSocketStream.h CFStream.h CodeFragments.h Collections.h Components.h DateTimeUtils.h Debugging.h Endian.h Files.h Finder.h FinderRegistry.h FixMath.h Folders.h Gestalt.h HFSVolumes.h IOMacOSTypes.h KeychainCore.h KeychainHI.h LSInfo.h LSOpen.h LowMem.h MDImporter.h MDItem.h MDQuery.h MDSchema.h MacErrors.h MacLocales.h MacMemory.h Math64.h MixedMode.h Multiprocessing.h MultiprocessingInfo.h NumberFormatting.h OSTypes.h OSUtils.h

OpenTransport.h
OpenTransportProtocol.h
OpenTransportProviders.h
PEFBinaryFormat.h
PLStringFuncs.h
Power.h
QuickTimeComponents.k.h
Resources.h
SCSI.h
SKAnalysis.h
SKDocument.h
SKIndex.h
SKSearch.h
SKSummary.h
Script.h
StringCompare.h
TextCommon.h
TextEncodingConverter.h
TextEncodingPlugin.h
TextUtils.h
Threads.h
Timer.h
ToolUtils.h
TypeSelect.h
UTCUtils.h
UnicodeConverter.h
UnicodeUtilities.h
fenv.h
fp.h
pyport.h
queue.h
syslog.h
types.h

This collection of documents provides the API reference for the Core Services framework, which encompasses many fundamental operating system services used by Carbon applications.

Opaque Types

CFFTPStream Reference

Derived From:	CFType
Framework:	CoreServices
Declared in	CFNetwork/CFFTPStream.h
Companion guide	CFNetwork Programming Guide

Overview

This document describes the CFStream functions for working with FTP connections. It is part of the CFFTP API.

Functions

CFFTPCreateParsedResourceListing

Parses an FTP listing to a dictionary.

```
CFIndex CFFTPCreateParsedResourceListing (
   CFAllocatorRef alloc,
   const UInt8 *buffer,
   CFIndex bufferLength,
   CFDictionaryRef *parsed
);
```

Parameters

alloc

The allocator to use to allocate memory for the dictionary. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

buffer

A pointer to a buffer holding zero or more lines of resource listing.

bufferLength

The length in bytes of the buffer pointed to by `buffer`.

parsed

Upon return, contains a dictionary containing the parsed resource information. If parsing fails, a `NULL` pointer is returned.

Return Value

The number of bytes parsed, `0` if no bytes were available for parsing, or `-1` if parsing failed.

Discussion
This function examines the contents of buffer as an FTP directory listing and parses into a CFDictionary the information for a single file or folder. The CFDictionary is returned in the `parsed` parameter, and the number of bytes used from buffer is returned.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CFFTPStream.h`

CFReadStreamCreateWithFTPURL

Creates an FTP read stream.

```
CFReadStreamRef CFReadStreamCreateWithFTPURL (
   CFAllocatorRef alloc,
   CFURLRef ftpURL
);
```

Parameters

`alloc`

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

`ftpURL`

A pointer to a CFURL structure for the URL to be downloaded that can be created by calling any of the `CFURLCreate` functions, such as `CFURLCreateWithString`.

Return Value
A new read stream, or `NULL` if the call failed. Ownership follows the Create Rule.

Discussion
This function creates an FTP read stream for downloading data from an FTP URL. If the `ftpURL` parameter is created with the user name and password as part of the URL (such as `ftp://username:password@ftp.example.com`) then the user name and password will automatically be set in the `CFReadStream`. Otherwise, call `CFReadStreamSetProperty` to set the steam's properties, such as `kCFStreamPropertyFTPUserName` and `kCFStreamPropertyFTPPassword` to associate a user name and password with the stream that are used to log in when the stream is opened. See "Constants" (page 21) for a description of all FTP stream properties.

To initiate a connection with the FTP server, call `CFReadStreamOpen`. To read the FTP stream, call `CFReadStreamRead`. If the URL refers to a directory, the stream provides the listing results sent by the server. If the URL refers to a file, the stream provides the data in that file.

To close a connection with the FTP server, call `CFReadStreamClose`.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CFFTPStream.h`

CFWriteStreamCreateWithFTPURL

Creates an FTP write stream.

```
CFWriteStreamRef CFWriteStreamCreateWithFTPURL (
   CFAllocatorRef alloc,
   CFURLRef ftpURL
);
```

Parameters

alloc

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

ftpURL

A pointer to a CFURL structure for the URL to be uploaded created by calling any of the `CFURLCreate` functions, such as `CFURLCreateWithString`.

Return Value
A new write stream, or `NULL` if the call failed. Ownership follows the Create Rule.

Discussion
This function creates an FTP write stream for uploading data to an FTP URL. If the `ftpURL` parameter is created with the user name and password as part of the URL (such as `ftp://username:password@ftp.example.com`) then the user name and password will automatically be set in the `CFWriteStream`. Call `CFWriteStreamSetProperty` to set the steam's properties, such as `kCFStreamPropertyFTPUserName` and `kCFStreamPropertyFTPPassword` to associate a user name and password with the stream that are used to log in when the stream is opened. See "Constants" (page 21) for a description of all FTP stream properties.

After creating the write stream, you can call `CFWriteStreamGetStatus` at any time to check the status of the stream.

To initiate a connection with the FTP server, call `CFWriteStreamOpen`. If the URL specifies a directory, the open is immediately followed by the event `kCFStreamEventEndEncountered` (and the stream passes to the state `kCFStreamStatusAtEnd`). Once the stream reaches this state, the directory has been created. Intermediary directories are not created.

To write to the FTP stream, call `CFWriteStreamWrite`.

To close a connection with the FTP server, call `CFWriteStreamClose`.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CFFTPStream.h`

Constants

CFStream FTP Property Constants

Constants for setting and copying CFStream FTP properties.

```
const CFStringRef kCFStreamPropertyFTPUserName;
const CFStringRef kCFStreamPropertyFTPPassword;
const CFStringRef kCFStreamPropertyFTPUsePassiveMode;
const CFStringRef kCFStreamPropertyFTPResourceSize;
const CFStringRef kCFStreamPropertyFTPFetchResourceInfo;
const CFStringRef kCFStreamPropertyFTPFileTransferOffset;
const CFStringRef kCFStreamPropertyFTPAttemptPersistentConnection;
const CFStringRef kCFStreamPropertyFTPProxy;
const CFStringRef kCFStreamPropertyFTPProxyHost;
extern const CFStringRef kCFStreamPropertyFTPProxyPort;
extern const CFStringRef kCFStreamPropertyFTPProxyUser;
extern const CFStringRef kCFStreamPropertyFTPProxyPassword;
```

Constants

`kCFStreamPropertyFTPUserName`

FTP User Name stream property key for set and copy operations. A value of type CFString for storing the login user name. Don't set this property when anonymous FTP is desired.

Available in Mac OS X v10.3 and later.

Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPPassword`

FTP Password stream property key for set and copy operations. A value of type CFString for storing the login password. Don't set this property when anonymous FTP is desired.

Available in Mac OS X v10.3 and later.

Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPUsePassiveMode`

FTP Passive Mode stream property key for set and copy operations. Set this property to `kCFBooleanTrue` to enable passive mode; set this property to `kCFBooleanFalse` to disable passive mode.

Available in Mac OS X v10.3 and later.

Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPResourceSize`

FTP Resource Size read stream property key copy operations. This property stores a CFNumber of type `kCFNumberLongLongType` representing the size of a resource in bytes.

Available in Mac OS X v10.3 and later.

Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPFetchResourceInfo`

FTP Fetch Resource Information stream property key for set and copy operations. Set this property to `kCFBooleanTrue` to require that resource information, such as size, must be provided before download starts; set this property to `kCFBooleanFalse` to allow downloads to start without resource information. For this version, size is the only resource information.

Available in Mac OS X v10.3 and later.

Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPFileTransferOffset`

FTP File Transfer Offset stream property key for set and copy operations. The value of this property is a CFNumber of type `kCFNumberLongLongType` representing the file offset at which to start the transfer.

Available in Mac OS X v10.3 and later.

Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPAttemptPersistentConnection`
: FTP Attempt Persistent Connection stream property key for set and copy operations. Set this property to `kCFBooleanTrue` to enable the reuse of existing server connections; set this property to `kCFBooleanFalse` to not reuse existing server connections. By default, this property is set to `kCFBooleanTrue`.

  Available in Mac OS X v10.3 and later.

  Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPProxy`
: FTP Proxy stream property key for set and copy operations. The property is a value of type CFDictionary that holds proxy dictionary key-value pairs. The dictionary returned by SystemConfiguration can also be set as the value of this property.

  Available in Mac OS X v10.3 and later.

  Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPProxyHost`
: FTP Proxy Host stream property key or an FTP Proxy dictionary key for set and copy operations. The value of this property is a CFString containing the host name of a proxy server. This property can be set and copied individually or via a CFDictionary. This property is the same as the `kSCPropNetProxiesFTPProxy` property defined in `SCSchemaDefinitions.h`.

  Available in Mac OS X v10.3 and later.

  Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPProxyPort`
: FTP Proxy Port stream property key or an FTP Proxy dictionary key for set and copy operations. The value of this property is a CFNumber of type `kCFNumberIntType` containing the port number of a proxy server. This property can be set and copied individually or via a CFDictionary. This property is the same as the `kSCPropNetProxiesFTPPort` property defined in `SCSchemaDefinitions.h`.

  Available in Mac OS X v10.3 and later.

  Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPProxyUser`
: FTP Proxy Host stream property key or FTP Proxy dictionary key for set and copy operations. The value of this property is a CFString containing the username to be used when connecting to the proxy server.

  Available in Mac OS X v10.3 and later.

  Declared in `CFFTPStream.h`.

`kCFStreamPropertyFTPProxyPassword`
: FTP Proxy Port stream property key or FTP Proxy dictionary key for set and copy operations. The value of this property is a CFString containing the password to be used when connecting to the proxy server.

  Available in Mac OS X v10.3 and later.

  Declared in `CFFTPStream.h`.

Discussion
The CFStream property constants are used to specify the property to set when calling `CFReadStreamSetProperty` or `CFWriteStreamSetProperty` and to copy when calling `CFReadStreamCopyProperty` or `CFWriteStreamCopyProperty`. They can also be passed to a CFDictionary creator or to an item accessor or mutator.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CFNetwork/CFFTPStream.h`

CFStream FTP Resource Constants

FTP resource constants.

```
const CFStringRef kCFFTPResourceMode;
const CFStringRef kCFFTPResourceName;
const CFStringRef kCFFTPResourceOwner;
const CFStringRef kCFFTPResourceGroup;
const CFStringRef kCFFTPResourceLink;
const CFStringRef kCFFTPResourceSize;
const CFStringRef kCFFTPResourceType;
const CFStringRef kCFFTPResourceModDate;
```

Constants

`kCFFTPResourceMode`
: CFDictionary key for getting the CFNumber containing the access permissions, defined in `sys/types.h`, of the FTP resource.

  Available in Mac OS X version 10.3 and later.

  Declared in `CFFTPStream.h`.

`kCFFTPResourceName`
: CFDictionary key for getting the CFString containing the name of the FTP resource.

  Available in Mac OS X version 10.3 and later.

  Declared in `CFFTPStream.h`.

`kCFFTPResourceOwner`
: CFDictionary key for getting the CFString containing the name of the owner of the FTP resource.

  Available in Mac OS X version 10.3 and later.

  Declared in `CFFTPStream.h`.

`kCFFTPResourceGroup`
: CFDictionary key for getting the CFString containing the name of a group that shares the FTP resource.

  Available in Mac OS X version 10.3 and later.

  Declared in `CFFTPStream.h`.

`kCFFTPResourceLink`
: CFDictionary key for getting the CFString containing the symbolic link information. If the item is a symbolic link, the CFString contains the path to the item that the link references.

  Available in Mac OS X version 10.3 and later.

  Declared in `CFFTPStream.h`.

`kCFFTPResourceSize`
: CFDictionary key for getting the CFNumber containing the size in bytes of the FTP resource.

  Available in Mac OS X version 10.3 and later.

  Declared in `CFFTPStream.h`.

`kCFFTPResourceType`
: CFDictionary key for getting the CFNumber containing the type of the FTP resource as defined in `sys/dirent.h`.

  Available in Mac OS X version 10.3 and later.

  Declared in `CFFTPStream.h`.

`kCFFTPResourceModDate`
: CFDictionary key for getting the CFDate containing the last date and time the FTP resource was modified.

  Available in Mac OS X version 10.3 and later.

  Declared in `CFFTPStream.h`.

Discussion
The values of FTP resource keys are extracted from a line of the directory list by the `CFFTPCreateParsedResourceListing` (page 19) function.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CFNetwork/CFFTPStream.h`

Error Domains

Error domains specific to `CFFTPStream` calls.

```
extern const SInt32 kCFStreamErrorDomainFTP;
```

Constants
`kCFStreamErrorDomainFTP`
: Error domain that returns the last result code returned by the FTP server.

  Available in Mac OS X version 10.3 and later.

  Declared in `CFFTPStream.h`.

Discussion
To determine the source of an error, examine the `userInfo` dictionary included in the `CFError` object returned by a function call or call `CFErrorGetDomain` and pass in the `CFError` object and the domain whose value you want to read.

CFHost Reference

Derived From:	CFType
Framework:	CoreServices
Declared in	CFNetwork/CFHost.h
Companion guide	CFNetwork Programming Guide

Overview

The CFHost API allows you to create instances of the CFHost object that you can use to acquire host information, including names, addresses, and reachability information.

The process of acquiring information about a host is known as resolution. Begin by calling `CFHostCreateWithAddress` or `CFHostCreateWithName` to create an instance of a CFHost using an address or a name, respectively. If you want to resolve the host asynchronously. call `CFHostSetClient` to associate your client context and user-defined callback function with the host. Then call `CFHostScheduleWithRunLoop` to schedule the host on a run loop.

To start resolution, call `CFHostStartInfoResolution`. If you set up for asynchronous resolution, `CFHostStartInfoResolution` returns immediately. Your callback function will be called when resolution is complete. If you didn't set up for asynchronous resolution, `CFHostStartInfoResolution` blocks until resolution is complete, an error occurs, or the resolution is cancelled.

When resolution is complete, call `CFHostGetAddressing` or `CFHostGetNames` to get an array of known addresses or known names, respectively, for the host. Call `CFHostGetReachability` to get flags, declared in `SystemConfiguration/SCNetwork.h`, that describe the reachability of the host.

When you no longer need a CFHost object, call `CFHostUnscheduleFromRunLoop` and `CFHostSetClient` to disassociate the host from your user-defined client context and callback function (if it was set up for asynchronous resolution). Then dispose of it.

Functions by Task

Creating a host

`CFHostCreateCopy` (page 29)
Creates a new host object by copying.

`CFHostCreateWithAddress` (page 29)
Uses an address to create an instance of a host object.

`CFHostCreateWithName` (page 30)
Uses a name to create an instance of a host object.

CFHost Functions

`CFHostCancelInfoResolution` (page 28)
Cancels the resolution of a host.

`CFHostGetAddressing` (page 31)
Gets the addresses from a host.

`CFHostGetNames` (page 31)
Gets the names from a CFHost.

`CFHostGetReachability` (page 32)
Gets reachability information from a host.

`CFHostStartInfoResolution` (page 34)
Starts resolution for a host object.

`CFHostSetClient` (page 34)
Associates a client context and a callback function with a CFHost object or disassociates a client context and callback function that were previously set.

`CFHostScheduleWithRunLoop` (page 33)
Schedules a CFHost on a run loop.

`CFHostUnscheduleFromRunLoop` (page 35)
Unschedules a CFHost from a run loop.

Getting the CFHost Type ID

`CFHostGetTypeID` (page 33)
Gets the Core Foundation type identifier for the CFHost opaque type.

Functions

CFHostCancelInfoResolution

Cancels the resolution of a host.

```
void CFHostCancelInfoResolution (
   CFHostRef theHost,
   CFHostInfoType info
);
```

Parameters

theHost
The host for which a resolution is to be cancelled. This value must not be `NULL`.

info

A value of type `CFHostInfoType` specifying the type of resolution that is to be cancelled. See CFHostInfoType Constants (page 38) for possible values.

Discussion
This function cancels the asynchronous or synchronous resolution specified by `info` for the host specified by `theHost`.

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CFHost.h`

CFHostCreateCopy

Creates a new host object by copying.

```
CFHostRef CFHostCreateCopy (
   CFAllocatorRef alloc,
   CFHostRef host
);
```

Parameters

alloc

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

addr

The host to copy. This value must not be `NULL`.

Return Value
A valid CFHost object or `NULL` if the copy could not be created. The new host contains a copy of all previously resolved data from the original host. Ownership follows the Create Rule.

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CFHost.h`

CFHostCreateWithAddress

Uses an address to create an instance of a host object.

```
CFHostRef CFHostCreateWithAddress (
   CFAllocatorRef allocator,
   CFDataRef addr
);
```

Parameters

alloc

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

addr

A CFDataRef object containing a `sockaddr` structure for the address of the host. This value must not be `NULL`.

Return Value

A valid CFHostRef object that can be resolved, or `NULL` if the host could not be created. Ownership follows the Create Rule.

Discussion

Call `CFHostStartInfoResolution` (page 34) to resolve the return object's name and reachability information.

Special Considerations

This function is thread safe.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CFHost.h`

CFHostCreateWithName

Uses a name to create an instance of a host object.

```
CFHostRef CFHostCreateWithName (
   CFAllocatorRef allocator,
   CFStringRef hostname
);
```

Parameters

alloc

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

hostname

A string representing the name of the host. This value must not be `NULL`.

Return Value

A valid CFHostRef object that can be resolved, or `NULL` if the host could not be created. Ownership follows the Create Rule.

Discussion

Call `CFHostStartInfoResolution` (page 34) to resolve the object's addresses and reachability information.

Special Considerations

This function is thread safe.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CFHost.h`

CFHostGetAddressing

Gets the addresses from a host.

```
CFArrayRef CFHostGetAddressing (
   CFHostRef theHost,
   Boolean *hasBeenResolved
);
```

Parameters

`theHost`

The CFHost whose addresses are to be obtained. This value must not be `NULL`.

`hasBeenResolved`

On return, a pointer to a Boolean that is `TRUE` if addresses were available and `FALSE` if addresses were not available. This parameter can be null.

`function result`

A CFArray of addresses where address is a `sockaddr` structure wrapped by a CFDataRef, or null if no addresses were available.

Discussion

This function gets the addresses from a CFHost. The CFHost must have been previously resolved. To resolve a CFHost, call `CFHostStartInfoResolution` (page 34).

Special Considerations

This function gets the addresses in a thread-safe way, but the resulting data is not thread-safe. The data is returned as a "get" as opposed to a copy, so the data is not safe if the CFHost is altered from another thread.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CFHost.h`

CFHostGetNames

Gets the names from a CFHost.

```
CFArrayRef CFHostGetNames (
   CFHostRef theHost,
   Boolean *hasBeenResolved
);
```

Parameters

theHost

The host to examine. The host must have been previously resolved. (To resolve a host, call `CFHostStartInfoResolution` (page 34).) This value must not be `NULL`.

hasBeenResolved

On return, contains `TRUE` if names were available, otherwise `FALSE`. This value may be `NULL`.

Return Value

An array containing the of names of *theHost*, or `NULL` if no names were available.

Special Considerations

This function gets the names in a thread-safe way, but the resulting data is not thread-safe. The data is returned as a "get" as opposed to a copy, so the data is not safe if the CFHost is altered from another thread.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CFHost.h`

CFHostGetReachability

Gets reachability information from a host.

```
CFDataRef CFHostGetReachability (
   CFHostRef theHost,
   Boolean *hasBeenResolved
);
```

Parameters

theHost

The host whose reachability is to be obtained. The host must have been previously resolved. (To resolve a host, call `CFHostStartInfoResolution` (page 34).) This value must not be `NULL`.

hasBeenResolved

On return, contains `TRUE` if the reachability was available, otherwise `FALSE`. This value may be `NULL`.

Return Value

A CFData object that wraps the reachability flags (`SCNetworkConnectionFlags`) defined in `SystemConfiguration/SCNetwork.h`, or `NULL` if reachability information was not available.

Special Considerations

This function gets reachability information in a thread-safe way, but the resulting data is not thread-safe. The data is returned as a "get" as opposed to a copy, so the data is not safe if the CFHost is altered from another thread.

Availability

Available in Mac OS X version 10.3 and later.

Declared In
`CFHost.h`

CFHostGetTypeID

Gets the Core Foundation type identifier for the CFHost opaque type.

```
CFTypeID CFHostGetTypeID ();
```

Return Value
The Core Foundation type identifier for the CFHost opaque type.

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CFHost.h`

CFHostScheduleWithRunLoop

Schedules a CFHost on a run loop.

```
void CFHostScheduleWithRunLoop (
   CFHostRef theHost,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

theHost
The host to be schedule on a run loop. This value must not be `NULL`.

runLoop
The run loop on which to schedule *theHost*. This value must not be `NULL`.

runLoopMode
The mode on which to schedule *theHost*. This value must not be `NULL`.

Discussion
Schedules *theHost* on a run loop, which causes resolutions of the host to be performed asynchronously. The caller is responsible for ensuring that at least one of the run loops on which the host is scheduled is being run.

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CFHost.h`

CFHostSetClient

Associates a client context and a callback function with a CFHost object or disassociates a client context and callback function that were previously set.

```
Boolean CFHostSetClient (
   CFHostRef theHost,
   CFHostClientCallBack clientCB,
   CFHostClientContext *clientContext
);
```

Parameters

theHost

The host to modify. The value must not be `NULL`.

clientCB

The callback function to associate with *theHost*. The callback function will be called when a resolution completes or is cancelled. If you are calling this function to disassociate a client context and callback from `theHost`, p`clientCB`ass `NULL`.

clientContext

A `CFHostClientContext` (page 37) structure whose `info` field will be passed to the callback function specified by `clientCB` when `clientCB` is called. This value must not be `NULL` when setting an association.

Pass `NULL` when disassociating a client context and a callback from a host.

Return Value

`TRUE` if the association could be set or unset, otherwise `FALSE`.

Discussion

The callback function specified by `clientCB` will be called when a resolution completes or is cancelled.

Special Considerations

This function is thread safe.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CFHost.h`

CFHostStartInfoResolution

Starts resolution for a host object.

```
Boolean CFHostStartInfoResolution (
   CFHostRef theHost,
   CFHostInfoType info,
   CFStreamError *error
);
```

Parameters

theHost

The host, obtained by previously calling `CFHostCreateCopy` (page 29), `CFHostCreateWithAddress` (page 29), or `CFHostCreateWithName` (page 30), that is to be resolved. This value must not be `NULL`.

info
: A value of type `CFHostInfoType` specifying the type of information that is to be retrieved. See CFHostInfoType Constants (page 38) for possible values.

error
: A pointer to a `CFStreamError` structure, that, if an error occurs, is set to the error and the error's domain. In synchronous mode, the error indicates why resolution failed, and in asynchronous mode, the error indicates why resolution failed to start.

Return Value
`TRUE` if the resolution was started (asynchronous mode); `FALSE` if another resolution is already in progress for *theHost* or if an error occurred.

Discussion
This function retrieves the information specified by `info` and stores it in the host.

In synchronous mode, this function blocks until the resolution has completed, in which case this function returns `TRUE`, until the resolution is stopped by calling `CFHostCancelInfoResolution` (page 28) from another thread, in which case this function returns `FALSE`, or until an error occurs.

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CFHost.h`

CFHostUnscheduleFromRunLoop

Unschedules a CFHost from a run loop.

```
void CFHostUnscheduleFromRunLoop (
   CFHostRef theHost,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

theService
: The host to unschedule. This value must not be `NULL`.

runLoop
: The run loop. This value must not be `NULL`.

runLoopMode
: The mode from which the service is to be unscheduled. This value must not be `NULL`.

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CFHost.h`

Callbacks

CFHostClientCallBack

Defines a pointer to the callback function that is called when an asynchronous resolution of a CFHost completes or an error occurs for an asynchronous CFHost resolution.

```
typedef void (CFHostClientCallBack) (
   CFHostRef theHost,
   CFHostInfoType typeInfo,
   const CFStreamError *error,
   void *info);
```

If you name your callback `MyHostClientCallBack`, you would declare it like this:

```
void MyHostClientCallBack (
   CFHostRef theHost,
   CFHostInfoType typeInfo,
   const CFStreamError *error,
   void *info
);
```

Parameters

`theHost`
The host for which an asynchronous resolution has been completed.

`typeInfo`
Value of type `CFHostInfoType` representing the type of information (addresses, names, or reachability information) obtained by the completed resolution. See CFHostInfoType Constants (page 38) for possible values.

`error`
If the resolution failed, contains a `CFStreamError` structure whose `error` field contains an error code.

`info`
User-defined context information. The value pointed to by `info` is the same as the value pointed to by the `info` field of the `CFHostClientContext` (page 37) structure that was provided when the host was associated with this callback function.

Discussion
The callback function for a CFHost object is called one or more times when an asynchronous resolution completes for the specified host, when an asynchronous resolution is cancelled, or when an error occurs during an asynchronous resolution.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CFHost.h`

Data Types

CFHostRef

An opaque reference representing an CFHost object.

```
typedef struct __CFHost* CFHostRef;
```

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CFHost.h`

CFHostClientContext

A structure containing user-defined data and callbacks for CFHost objects.

```
struct CFHostClientContext {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
} CFHostClientContext;
typedef struct CFHostClientContext CFHostClientContext;
```

Fields

`version`
: The version number of the structure type passed as a parameter to the host client function. The only valid version number is `0`.

`info`
: An arbitrary pointer to allocated memory containing user-defined data that can be associated with the host and that is passed to the callbacks.

`retain`
: The callback used to add a retain for the host on the info pointer for the life of the host, and may be used for temporary references the host needs to take. This callback returns the actual info pointer to store in the host, almost always just the pointer passed as the parameter.

`release`
: The callback used to remove a retain previously added for the host on the info pointer.

`copyDescription`
: The callback used to create a descriptive string representation of the info pointer (or the data pointed to by the info pointer) for debugging purposes. This callback is called by the `CFCopyDescription` function.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`CFHost.h`

Constants

CFHostInfoType Constants

Values indicating the type of data that is to be resolved or the type of data that was resolved.

```
enum CFHostInfoType {
    kCFHostAddresses = 0,
    kCFHostNames = 1,
    kCFHostReachability = 2
};
typedef enum CFHostInfoType CFHostInfoType;
```

Constants

`kCFHostAddresses`

Specifies that addresses are to be resolved or that addresses were resolved.

Available in Mac OS X v10.3 and later.

Declared in `CFHost.h`.

`kCFHostNames`

Specifies that names are to be resolved or that names were resolved.

Available in Mac OS X v10.3 and later.

Declared in `CFHost.h`.

`kCFHostReachability`

Specifies that reachability information is to be resolved or that reachability information was resolved.

Available in Mac OS X v10.3 and later.

Declared in `CFHost.h`.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`CFNetwork/CFHost.h`

Error Domains

Error domains specific to `CFHost` calls.

```
extern const SInt32 kCFStreamErrorDomainNetDB;
extern const SInt32 kCFStreamErrorDomainSystemConfiguration;
```

Constants

`kCFStreamErrorDomainNetDB`

The error domain that returns errors from the network database (DNS resolver) layer (described in `/usr/include/netdb.h`).

Available in Mac OS X version 10.5 and later.

Declared in `CFHost.h`.

`kCFStreamErrorDomainSystemConfiguration`

The error domain that returns errors from the system configuration layer (described in *System Configuration Framework Reference*).

Available in Mac OS X version 10.5 and later.

Declared in `CFHost.h`.

Discussion

To determine the source of an error, examine the `userInfo` dictionary included in the `CFError` object returned by a function call or call `CFErrorGetDomain` and pass in the `CFError` object and the domain whose value you want to read.

CFHTTPAuthentication Reference

Derived From:	CFType
Framework:	CoreServices
Declared in	CFNetwork/CFHTTPAuthentication.h
Companion guide	CFNetwork Programming Guide

Overview

The CFHTTPAuthentication opaque type provides an abstraction of HTTP authentication information.

Functions by Task

Creating an HTTP authentication

`CFHTTPAuthenticationCreateFromResponse` (page 44)
Uses an authentication failure response to create a CFHTTPAuthentication object.

CFHTTP Authentication Functions

This section describes the CFNetwork authentication functions that are used to manage authentication information associated with a request. The functions work with a CFHTTPAuthentication object, which is created from an HTTP response that failed with a 401 or 407 error code.

When you have analyzed the CFHTTPAuthentication object and acquired the necessary credentials to perform the authentication, call `CFHTTPMessageApplyCredentials` (page 55) or `CFHTTPMessageApplyCredentialDictionary` (page 54) to perform the authentication.

`CFHTTPAuthenticationAppliesToRequest` (page 42)
Returns a Boolean value that indicates whether a CFHTTPAuthentication object is associated with a CFHTTPMessage object.

`CFHTTPAuthenticationCopyDomains` (page 43)
Returns an array of domain URLs to which a given CFHTTPAuthentication object can be applied.

`CFHTTPAuthenticationCopyMethod` (page 43)
Gets the strongest authentication method that will be used when a CFHTTPAuthentication object is applied to a request.

`CFHTTPAuthenticationCopyRealm` (page 43)
Gets an authentication information's namespace.

`CFHTTPAuthenticationIsValid` (page 45)
Returns a Boolean value that indicates whether a CFHTTPAuthentication object is valid.

`CFHTTPAuthenticationRequiresAccountDomain` (page 46)
Returns a Boolean value that indicates whether a CFHTTPAuthentication object uses an authentication method that requires an account domain.

`CFHTTPAuthenticationRequiresOrderedRequests` (page 46)
Returns a Boolean value that indicates whether authentication requests should be made one at a time.

`CFHTTPAuthenticationRequiresUserNameAndPassword` (page 47)
Returns a Boolean value that indicates whether a CFHTTPAuthentication object uses an authentication method that requires a username and a password.

Getting the CFHTTPAuthentication type ID

`CFHTTPAuthenticationGetTypeID` (page 45)
Gets the Core Foundation type identifier for the CFHTTPAuthentication opaque type.

Functions

CFHTTPAuthenticationAppliesToRequest

Returns a Boolean value that indicates whether a CFHTTPAuthentication object is associated with a CFHTTPMessage object.

```
Boolean CFHTTPAuthenticationAppliesToRequest (
   CFHTTPAuthenticationRef auth,
   CFHTTPMessageRef request
);
```

Parameters

auth
The CFHTTPAuthentication object to examine.

request
Request that `auth` is to be tested against.

Return Value
`TRUE` if `auth` is associated with `request`, otherwise `FALSE`.

Discussion
If this function returns `TRUE`, you can use *auth* to provide authentication information when using *request*.

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

Declared In
`CFHTTPAuthentication.h`

CFHTTPAuthenticationCopyDomains

Returns an array of domain URLs to which a given CFHTTPAuthentication object can be applied.

```
CFArrayRef CFHTTPAuthenticationCopyDomains (
   CFHTTPAuthenticationRef auth
);
```

Parameters

auth

The CFHTTPAuthentication object to examine.

Return Value

A CFArray object that contains the domain URL's to which `auth` should be applied. Ownership follows the Create Rule.

Discussion

This function is provided for informational purposes only.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CFHTTPAuthentication.h`

CFHTTPAuthenticationCopyMethod

Gets the strongest authentication method that will be used when a CFHTTPAuthentication object is applied to a request.

```
CFStringRef CFHTTPAuthenticationCopyMethod (
   CFHTTPAuthenticationRef auth
);
```

Parameters

auth

The CFHTTPAuthentication object to examine.

Return Value

A string containing the authentication method that will be used *auth* is applied to a request. If more than one authentication method is available, the strongest authentication method is returned. Ownership follows the Create Rule.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CFHTTPAuthentication.h`

CFHTTPAuthenticationCopyRealm

Gets an authentication information's namespace.

```
CFStringRef CFHTTPAuthenticationCopyRealm (
   CFHTTPAuthenticationRef auth
);
```

Parameters

auth

The CFHTTPAuthentication object to examine.

Return Value
The namespace, if there is one; otherwise `NULL`. Ownership follows the Create Rule.

Discussion
Some authentication methods provide a namespace, and it is usually used to prompt the user for a name and password.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFHTTPAuthentication.h`

CFHTTPAuthenticationCreateFromResponse

Uses an authentication failure response to create a CFHTTPAuthentication object.

```
CFHTTPAuthenticationRef CFHTTPAuthenticationCreateFromResponse (
   CFAllocatorRef alloc,
   CFHTTPMessageRef response
);
```

Parameters

alloc

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

response

Response indicating an authentication failure; usually a 401 or a 407 response.

Return Value
CFHTTPAuthentication object that can be used for adding credentials to future requests. Ownership follows the Create Rule.

Discussion
This function uses a response containing authentication failure information to create a reference to a CFHTTPAuthentication object. You can use the object to add credentials to future requests. You can query the object to get the following information:

- whether it can be used and re-used to authenticate with its corresponding server [`CFHTTPAuthenticationIsValid` (page 45)]
- the authentication method that will be used when it is used to perform an authentication [`CFHTTPAuthenticationCopyMethod` (page 43)]
- whether it is associated with a particular CFHTTPMessageRef [`CFHTTPAuthenticationAppliesToRequest` (page 42)

- whether a user name and a password will be required when it is used to perform an authentication [CFHTTPAuthenticationRequiresUserNameAndPassword (page 47)]
- whether an account domain will be required when it is used to perform an authentication [CFHTTPAuthenticationRequiresAccountDomain (page 46)]
- whether authentication requests should be sent one at a time to the corresponding server [CFHTTPAuthenticationRequiresOrderedRequests (page 46)]
- the namespace (if any) that the domain uses to prompt for a name and password [CFHTTPAuthenticationCopyRealm (page 43)]
- the domain URLs the instance applies to [CFHTTPAuthenticationCopyDomains (page 43)]

When you have determined what information will be needed to perform the authentication and accumulated that information, call CFHTTPMessageApplyCredentials (page 55) or CFHTTPMessageApplyCredentialDictionary (page 54) to perform the authentication.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
CFHTTPAuthentication.h

CFHTTPAuthenticationGetTypeID

Gets the Core Foundation type identifier for the CFHTTPAuthentication opaque type.

```
CFTypeID CFHTTPAuthenticationGetTypeID ();
```

Return Value
The Core Foundation type identifier for the CFHTTPAuthentication opaque type.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
CFHTTPAuthentication.h

CFHTTPAuthenticationIsValid

Returns a Boolean value that indicates whether a CFHTTPAuthentication object is valid.

```
Boolean CFHTTPAuthenticationIsValid (
   CFHTTPAuthenticationRef auth,
   CFStreamError *error
);
```

Parameters

auth
The CFHTTPAuthentication object to examine.

error
Pointer to a CFStreamError structure, whose fields, if an error has occurred, are set to the error and the error's domain.

Return Value
`TRUE` if `auth` contains enough information to be applied to a request.

If this function returns `FALSE`, the CFHTTPAuthentication object may still contain useful information, such as the name of an unsupported authentication method.

Discussion
If this function returns `TRUE` for *`auth`*, the object is good for use with functions such as `CFHTTPMessageApplyCredentials` (page 55) and `CFHTTPMessageApplyCredentialDictionary` (page 54). If this function returns `FALSE`, *`auth`* is invalid, and authentications using it will not succeed.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFHTTPAuthentication.h`

CFHTTPAuthenticationRequiresAccountDomain

Returns a Boolean value that indicates whether a CFHTTPAuthentication object uses an authentication method that requires an account domain.

```
Boolean CFHTTPAuthenticationRequiresAccountDomain (
   CFHTTPAuthenticationRef auth
);
```

Parameters
`auth`
The CFHTTPAuthentication object to examine.

Return Value
`TRUE` if `auth` uses an authentication method that requires an account domain, otherwise `FALSE`.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFHTTPAuthentication.h`

CFHTTPAuthenticationRequiresOrderedRequests

Returns a Boolean value that indicates whether authentication requests should be made one at a time.

```
Boolean CFHTTPAuthenticationRequiresOrderedRequests (
   CFHTTPAuthenticationRef auth
);
```

Parameters
`auth`
The CFHTTPAuthentication object to examine.

Return Value
`TRUE` if `auth` requires ordered requests, otherwise `FALSE`.

Discussion
Some authentication methods require that future requests must be performed in an ordered manner. If this function returns `TRUE`, clients can improve their chances of authenticating successfully by issuing requests one at a time as responses come back from the server.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFHTTPAuthentication.h`

CFHTTPAuthenticationRequiresUserNameAndPassword

Returns a Boolean value that indicates whether a CFHTTPAuthentication object uses an authentication method that requires a username and a password.

```
Boolean CFHTTPAuthenticationRequiresUserNameAndPassword (
   CFHTTPAuthenticationRef auth
);
```

Parameters

auth

The CFHTTPAuthentication object to examine.

Return Value
`TRUE` if `auth` requires a username and password when it is applied to a request; otherwise, `FALSE`.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFHTTPAuthentication.h`

Data Types

CFHTTPAuthenticationRef

An opaque reference representing HTTP authentication information.

```
typedef struct __CFHTTPAuthentication *CFHTTPAuthenticationRef;
```

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFHTTPAuthentication.h`

Constants

CFHTTP Authentication Scheme Constants

Specifies the authentication scheme when adding authentication information to a CFHTTP request message object.

```
const CFStringRef kCFHTTPAuthenticationSchemeBasic;
const CFStringRef kCFHTTPAuthenticationSchemeDigest;
const CFStringRef kCFHTTPAuthenticationSchemeNegotiate;
const CFStringRef kCFHTTPAuthenticationSchemeNTLM;
```

Constants

`kCFHTTPAuthenticationSchemeBasic`

Specifies basic authentication consisting of a user name and a password.

Available in Mac OS X v10.1 and later.

Declared in `CFHTTPMessage.h`.

`kCFHTTPAuthenticationSchemeDigest`

Reserved.

Available in Mac OS X v10.1 and later.

Declared in `CFHTTPMessage.h`.

`kCFHTTPAuthenticationSchemeNegotiate`

Specifies the Negotiate authentication scheme.

Available in Mac OS X v10.5 and later.

Declared in `CFHTTPMessage.h`.

`kCFHTTPAuthenticationSchemeNTLM`

Specifies the NTLM authentication scheme.

Available in Mac OS X v10.5 and later.

Declared in `CFHTTPMessage.h`.

Discussion

The authentication scheme constants are used to specify the authentication scheme when calling `CFHTTPMessageAddAuthentication` (page 53).

CFStream HTTP Authentication Error Constants

Authentication error codes that may be returned when trying to apply authentication to a request.

```
enum CFStreamErrorHTTPAuthentication{
    kCFStreamErrorHTTPAuthenticationTypeUnsupported = -1000,
    kCFStreamErrorHTTPAuthenticationBadUserName = -1001,
    kCFStreamErrorHTTPAuthenticationBadPassword = -1002
};
typedef enum CFStreamErrorHTTPAuthentication CFStreamErrorHTTPAuthentication;
```

Constants

`kCFStreamErrorHTTPAuthenticationTypeUnsupported`
Specified authentication type is not supported.

Available in Mac OS X v10.4 and later.

Declared in `CFHTTPAuthentication.h`.

`kCFStreamErrorHTTPAuthenticationBadUserName`
User name is in a format that is not suitable for the request. Currently, user names are decoded using `kCFStringEncodingISOLatin1`.

Available in Mac OS X v10.4 and later.

Declared in `CFHTTPAuthentication.h`.

`kCFStreamErrorHTTPAuthenticationBadPassword`
Password is in a format that is not suitable for the request. Currently, passwords are decoded using `kCFStringEncodingISOLatin1`.

Available in Mac OS X v10.4 and later.

Declared in `CFHTTPAuthentication.h`.

CFHTTPMessageApplyCredentialDictionary Keys

Constants for keys in the dictionary passed to `CFHTTPMessageApplyCredentialDictionary` (page 54).

```
const CFStringRef kCFHTTPAuthenticationUserName;
const CFStringRef kCFHTTPAuthenticationPassword;
const CFStringRef kCFHTTPAuthenticationAccountDomain;
```

Constants

`kCFHTTPAuthenticationUserName`
Username to use for authentication.

`kCFHTTPAuthenticationPassword`
Password to use for authentication.

Available in Mac OS X v10.4 and later.

Declared in `CFHTTPAuthentication.h`.

`kCFHTTPAuthenticationAccountDomain`
Account domain to use for authentication.

Available in Mac OS X v10.4 and later.

Declared in `CFHTTPAuthentication.h`.

CFHTTPMessage Reference

Derived From:	CFType
Framework:	CoreServices
Declared in	CFNetwork/CFHTTPMessage.h
Companion guides	Getting Started with Networking CFNetwork Programming Guide

Overview

The `CFHTTPMessage` opaque type represents an HTTP message.

Functions by Task

Creating a Message

`CFHTTPMessageCreateCopy` (page 60)
Gets a copy of a CFHTTPMessage object.

`CFHTTPMessageCreateEmpty` (page 60)
Creates and returns a new, empty `CFHTTPMessage` object.

`CFHTTPMessageCreateRequest` (page 61)
Creates and returns a `CFHTTPMessage` object for an HTTP request.

`CFHTTPMessageCreateResponse` (page 62)
Creates and returns a `CFHTTPMessage` object for an HTTP response.

Modifying a message

`CFHTTPMessageAppendBytes` (page 54)
Appends data to a `CFHTTPMessage` object.

`CFHTTPMessageSetBody` (page 64)
Sets the body of a `CFHTTPMessage` object.

`CFHTTPMessageSetHeaderFieldValue` (page 64)
Sets the value of a header field in an HTTP message.

Getting information from a message

`CFHTTPMessageCopyBody` (page 57)
Gets the body from a `CFHTTPMessage` object.

`CFHTTPMessageCopyAllHeaderFields` (page 56)
Gets all header fields from a `CFHTTPMessage` object.

`CFHTTPMessageCopyHeaderFieldValue` (page 57)
Gets the value of a header field from a `CFHTTPMessage` object.

`CFHTTPMessageCopyRequestMethod` (page 58)
Gets the request method from a `CFHTTPMessage` object.

`CFHTTPMessageCopyRequestURL` (page 58)
Gets the URL from a `CFHTTPMessage` object.

`CFHTTPMessageCopySerializedMessage` (page 59)
Serializes a CFHTTPMessage object.

`CFHTTPMessageCopyVersion` (page 59)
Gets the HTTP version from a `CFHTTPMessage` object.

`CFHTTPMessageIsRequest` (page 64)
Returns a boolean indicating whether the CFHTTPMessage is a request or a response.

`CFHTTPMessageIsHeaderComplete` (page 63)
Determines whether a message header is complete.

`CFHTTPMessageGetResponseStatusCode` (page 63)
Gets the status code from a `CFHTTPMessage` object representing an HTTP response.

`CFHTTPMessageCopyResponseStatusLine` (page 59)
Gets the status line from a `CFHTTPMessage` object.

Message authentication

`CFHTTPMessageApplyCredentials` (page 55)
Performs the authentication method specified by a `CFHTTPAuthentication` object.

`CFHTTPMessageApplyCredentialDictionary` (page 54)
Use a dictionary containing authentication credentials to perform the authentication method specified by a `CFHTTPAuthentication` object.

`CFHTTPMessageAddAuthentication` (page 53)
Adds authentication information to a request.

Getting the CFHTTPMessage type identifier

`CFHTTPMessageGetTypeID` (page 63)
Returns the Core Foundation type identifier for the `CFHTTPMessage` opaque type.

Functions

CFHTTPMessageAddAuthentication

Adds authentication information to a request.

```
Boolean CFHTTPMessageAddAuthentication (
   CFHTTPMessageRef request,
   CFHTTPMessageRef authenticationFailureResponse,
   CFStringRef username,
   CFStringRef password,
   CFStringRef authenticationScheme,
   Boolean forProxy
);
```

Parameters

request
The message to which to add authentication information.

authenticationFailureResponse
The response message that contains authentication failure information.

username
The username to add to the request.

password
The password to add to the request.

authenticationScheme
The authentication scheme to use (`kCFHTTPAuthenticationSchemeBasic`, `kCFHTTPAuthenticationSchemeNegotiate`, `kCFHTTPAuthenticationSchemeNTLM`, or `kCFHTTPAuthenticationSchemeDigest`), or pass `NULL` to use the strongest supported authentication scheme provided in the `authenticationFailureResponse` parameter.

forProxy
A flag indicating whether the authentication data that is being added is for a proxy's use (`TRUE`) or for a remote server's use (`FALSE`). If the error code provided by the `authenticationFailureResponse` parameter is 407, set `forProxy` to `TRUE`. If the error code is 401, set `forProxy` to `FALSE`.

Return Value

`TRUE` if the authentication information was successfully added, otherwise `FALSE`.

Discussion

This function adds the authentication information specified by the `username`, `password`, `authenticationScheme`, and `forProxy` parameters to the specified request message. The message referred to by the `authenticationFailureResponse` parameter typically contains a 401 or a 407 error code.

This function is best suited for sending a single request to the server. If you need to send multiple requests, use `CFHTTPMessageApplyCredentials` (page 55).

Availability

Available in Mac OS X version 10.1 and later.

Declared In

`CFHTTPMessage.h`

CFHTTPMessageAppendBytes

Appends data to a `CFHTTPMessage` object.

```
Boolean CFHTTPMessageAppendBytes (
   CFHTTPMessageRef message,
   const UInt8 *newBytes,
   CFIndex numBytes
);
```

Parameters

message
The message to modify.

newBytes
A reference to the data to append.

numBytes
The length of the data pointed to by `newBytes`.

Return Value

`TRUE` if the data was successfully appended, otherwise `FALSE`.

Discussion

This function appends the data specified by `newBytes` to the specified message object which was created by calling `CFHTTPMessageCreateEmpty` (page 60). The data is an incoming serialized HTTP request or response received from a client or a server. While appending the data, this function deserializes it, removes any HTTP-based formatting that the message may contain, and stores the message in the message object. You can then call `CFHTTPMessageCopyVersion` (page 59), `CFHTTPMessageCopyBody` (page 57), `CFHTTPMessageCopyHeaderFieldValue` (page 57), and `CFHTTPMessageCopyAllHeaderFields` (page 56) to get the message's HTTP version, the message's body, a specific header field, and all of the message's headers, respectively.

If the message is a request, you can also call `CFHTTPMessageCopyRequestURL` (page 58) and `CFHTTPMessageCopyRequestMethod` (page 58) to get the message's request URL and request method, respectively.

If the message is a response, you can also call `CFHTTPMessageGetResponseStatusCode` (page 63) and `CFHTTPMessageCopyResponseStatusLine` (page 59) to get the message's status code and status line, respectively.

Availability

Available in Mac OS X version 10.1 and later.

Declared In

`CFHTTPMessage.h`

CFHTTPMessageApplyCredentialDictionary

Use a dictionary containing authentication credentials to perform the authentication method specified by a `CFHTTPAuthentication` object.

```
Boolean CFHTTPMessageApplyCredentialDictionary (
   CFHTTPMessageRef request,
   CFHTTPAuthenticationRef auth,
   CFDictionaryRef dict,
   CFStreamError *error
);
```

Parameters

request

The request for which the authentication method is to be performed.

auth

A `CFHTTPAuthentication` object specifying the authentication method to perform.

dict

A dictionary containing authentication credentials to be applied to the request. For information on the keys in this dictionary, see `CFHTTPAuthenticationRef` (page 47).

error

If an error occurs, upon return contains a `CFStreamError` object that describes the error and the error's domain. Pass `NULL` if you don't want to receive error information.

Return Value

`TRUE` if the authentication was successful, otherwise, `FALSE`.

Discussion

This function performs the authentication method specified by `auth` on behalf of the request specified by `request` using the credentials contained in the dictionary specified by `dict`. The dictionary must contain values for the `kCFHTTPAuthenticationUsername` and `kCFHTTPAuthenticationPassword` keys. If `CFHTTPAuthenticationRequiresAccountDomain` (page 46) returns `TRUE` for *auth*, the dictionary must also contain a value for the `kCFHTTPAuthenticationAccountDomain` key.

Special Considerations

This function is thread safe as long as another thread does not alter the same `CFHTTPAuthentication` object at the same time.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CFHTTPAuthentication.h`

CFHTTPMessageApplyCredentials

Performs the authentication method specified by a `CFHTTPAuthentication` object.

```
Boolean CFHTTPMessageApplyCredentials (
   CFHTTPMessageRef request,
   CFHTTPAuthenticationRef auth,
   CFStringRef username,
   CFStringRef password,
   CFStreamError *error
);
```

Parameters

request

Request for which the authentication method is to be performed.

auth

A `CFHTTPAuthentication` object specifying the authentication method to perform.

username

Username for performing the authentication.

password

Password for performing the authentication.

error

If an error occurs, upon return contains a `CFStreamError` object that describes the error and the error's domain. Pass `NULL` if you don't want to receive error information.

Return Value

`TRUE` if the authentication was successful, otherwise, `FALSE`.

Discussion

This function performs the authentication method specified by `auth` on behalf of the request specified by `request` using the credentials specified by `username` and `password`. If, in addition to a username and password, you also need to specify an account domain, call `CFHTTPMessageApplyCredentialDictionary` (page 54) instead of this function.

This function is appropriate for performing several authentication requests. If you only need to make a single authentication request, consider using `CFHTTPMessageAddAuthentication` (page 53) instead.

Special Considerations

This function is thread safe as long as another thread does not alter the same `CFHTTPMessage` object at the same time.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CFHTTPAuthentication.h`

CFHTTPMessageCopyAllHeaderFields

Gets all header fields from a `CFHTTPMessage` object.

```
CFDictionaryRef CFHTTPMessageCopyAllHeaderFields (
   CFHTTPMessageRef message
);
```

Parameters

message

The message to examine.

Return Value

A `CFDictionary` object containing keys and values that are `CFString` objects, where the key is the header fieldname and the dictionary value is the header field's value. Returns `NULL` if the header fields could not be copied. Ownership follows the Create Rule.

Availability

Available in Mac OS X version 10.1 and later.

Declared In

`CFHTTPMessage.h`

CFHTTPMessageCopyBody

Gets the body from a `CFHTTPMessage` object.

```
CFDataRef CFHTTPMessageCopyBody (
   CFHTTPMessageRef message
);
```

Parameters

message

The message to examine.

Return Value

A `CFData` object or `NULL` if there was a problem creating the object or if the there is no message body. Ownership follows the Create Rule.

Availability

Available in Mac OS X version 10.1 and later.

Declared In

`CFHTTPMessage.h`

CFHTTPMessageCopyHeaderFieldValue

Gets the value of a header field from a `CFHTTPMessage` object.

```
CFStringRef CFHTTPMessageCopyHeaderFieldValue (
   CFHTTPMessageRef message,
   CFStringRef headerField
);
```

Parameters

message

The message to examine.

headerField
The header field to copy.

Return Value
A `CFString` object containing a copy of the field specified by *headerField*, or `NULL` if there was a problem creating the object of if the specified header does not exist. Ownership follows the Create Rule.

Availability
Available in Mac OS X version 10.1 and later.

Declared In
`CFHTTPMessage.h`

CFHTTPMessageCopyRequestMethod

Gets the request method from a `CFHTTPMessage` object.

```
CFStringRef CFHTTPMessageCopyRequestMethod (
   CFHTTPMessageRef request
);
```

Parameters
request
The message to examine. This must be a request message.

Return Value
A `CFString` object containing a copy of the message's request method, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Availability
Available in Mac OS X version 10.1 and later.

Declared In
`CFHTTPMessage.h`

CFHTTPMessageCopyRequestURL

Gets the URL from a `CFHTTPMessage` object.

```
CFURLRef CFHTTPMessageCopyRequestURL (
   CFHTTPMessageRef request
);
```

Parameters
request
The message to examine. This must be a request message.

Return Value
A `CFURLRef` object containing the URL or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Availability
Available in Mac OS X version 10.1 and later.

Declared In
CFHTTPMessage.h

CFHTTPMessageCopyResponseStatusLine

Gets the status line from a CFHTTPMessage object.

```
CFStringRef CFHTTPMessageCopyResponseStatusLine (
   CFHTTPMessageRef response
);
```

Parameters

response

The message to examine. This must be a response message.

Return Value
A string containing the message's status line, or NULL if there was a problem creating the object. The status line includes the message's protocol version and a success or error code. Ownership follows the Create Rule.

Availability
Available in Mac OS X version 10.1 and later.

Declared In
CFHTTPMessage.h

CFHTTPMessageCopySerializedMessage

Serializes a CFHTTPMessage object.

```
CFDataRef CFHTTPMessageCopySerializedMessage (
   CFHTTPMessageRef request
);
```

Parameters

request

The message to serialize.

Return Value
A CFData object containing the serialized message, or NULL if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
This function returns a copy of a CFHTTPMessage object in serialized format that is ready for transmission.

Availability
Available in Mac OS X version 10.1 and later.

Declared In
CFHTTPMessage.h

CFHTTPMessageCopyVersion

Gets the HTTP version from a CFHTTPMessage object.

```
CFStringRef CFHTTPMessageCopyVersion (
   CFHTTPMessageRef message
);
```

Parameters

message

The message to examine.

Return Value

A `CFString` object or `NULL`, if there was a problem creating the object. Ownership follows the Create Rule.

Availability

Available in Mac OS X version 10.1 and later.

Declared In

`CFHTTPMessage.h`

CFHTTPMessageCreateCopy

Gets a copy of a CFHTTPMessage object.

```
CFHTTPMessageRef CFHTTPMessageCreateCopy (
   CFAllocatorRef alloc,
   CFHTTPMessageRef message
);
```

Parameters

allocator

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

message

The message to copy.

Return Value

A `CFHTTPMessage` object, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion

This function returns a copy of a `CFHTTPMessage` object that you can modify, for example, by calling `CFHTTPMessageCopyHeaderFieldValue` (page 57) or by calling`CFHTTPMessageSetBody` (page 64). Then serialize the message by calling`CFHTTPMessageCopySerializedMessage` (page 59) and send the serialized message to a client or a server.

Availability

Available in Mac OS X version 10.1 and later.

Declared In

`CFHTTPMessage.h`

CFHTTPMessageCreateEmpty

Creates and returns a new, empty `CFHTTPMessage` object.

```
CFHTTPMessageRef CFHTTPMessageCreateEmpty (
   CFAllocatorRef alloc,
   Boolean isRequest
);
```

Parameters

allocator

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

isRequest

A flag that determines whether to create an empty message request or an empty message response. Pass `TRUE` to create an empty request message; pass `FALSE` to create an empty response message.

Return Value

A new `CFHTTPMessage` object or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion

Call `CFHTTPMessageAppendBytes` (page 54) to store an incoming, serialized HTTP request or response message in the empty message object.

Availability

Available in Mac OS X version 10.1 and later.

Declared In

`CFHTTPMessage.h`

CFHTTPMessageCreateRequest

Creates and returns a `CFHTTPMessage` object for an HTTP request.

```
CFHTTPMessageRef CFHTTPMessageCreateRequest (
   CFAllocatorRef alloc,
   CFStringRef requestMethod,
   CFURLRef url,
   CFStringRef httpVersion
);
```

Parameters

allocator

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

requestMethod

The request method for the request. Use any of the request methods allowed by the HTTP version specified by *httpVersion*.

url

The URL to which the request will be sent.

httpVersion

The HTTP version for this message. Pass `kCFHTTPVersion1_0` or `kCFHTTPVersion1_1`.

Return Value

A new `CFHTTPMessage` object, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
This function returns a `CFHTTPMessage` object that you can use to build an HTTP request. Continue building the request by calling`CFHTTPMessageSetBody` (page 64) to set the message's body. Call `CFHTTPMessageCopyHeaderFieldValue` (page 57) to set the message's headers.

If you are using a `CFReadStream` object to send the message, call `CFReadStreamCreateForHTTPRequest` to create a read stream for the request. If you are not using `CFReadStream`, call `CFHTTPMessageCopySerializedMessage` (page 59) to make the message ready for transmission by serializing it.

Availability
Available in Mac OS X version 10.1 and later.

Declared In
`CFHTTPMessage.h`

CFHTTPMessageCreateResponse

Creates and returns a `CFHTTPMessage` object for an HTTP response.

```
CFHTTPMessageRef CFHTTPMessageCreateResponse (
   CFAllocatorRef alloc,
   CFIndex statusCode,
   CFStringRef statusDescription,
   CFStringRef httpVersion
);
```

Parameters

allocator
The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

statusCode
The status code for this message response. The status code can be any of the status codes defined in section 6.1.1 of RFC 2616.

statusDescription
The description that corresponds to the status code. Pass NULL to use the standard description for the given status code, as found in RFC 2616.

httpVersion
The HTTP version for this message response. Pass `kCFHTTPVersion1_0` or `kCFHTTPVersion1_1`.

Return Value
A new `CFHTTPMessage` object, or `NULL` if there was a problem creating the object. Ownership follows the Create Rule.

Discussion
This function returns a `CFHTTPMessage` object that you can use to build an HTTP response. Continue building the response by calling`CFHTTPMessageSetBody` (page 64) to set the message's body. Call `CFHTTPMessageSetHeaderFieldValue` (page 64) to set the message's headers. Then call `CFHTTPMessageCopySerializedMessage` (page 59) to make the message ready for transmission by serializing it.

Availability
Available in Mac OS X version 10.1 and later.

Declared In
CFHTTPMessage.h

CFHTTPMessageGetResponseStatusCode

Gets the status code from a CFHTTPMessage object representing an HTTP response.

```
CFIndex CFHTTPMessageGetResponseStatusCode (
   CFHTTPMessageRef response
);
```

Parameters

response
The message to examine. This must be a response message.

function result
The status code as defined by RFC 2616, section 6.1.1.

Availability
Available in Mac OS X version 10.1 and later.

Declared In
CFHTTPMessage.h

CFHTTPMessageGetTypeID

Returns the Core Foundation type identifier for the CFHTTPMessage opaque type.

```
CFTypeID CFHTTPMessageGetTypeID ();
```

Return Value
The Core Foundation type identifier for the CFHTTPMessage opaque type.

Availability
Available in Mac OS X version 10.1 and later.

Declared In
CFHTTPMessage.h

CFHTTPMessageIsHeaderComplete

Determines whether a message header is complete.

```
Boolean CFHTTPMessageIsHeaderComplete (
   CFHTTPMessageRef message
);
```

Parameters

message
The message to verify.

function result
TRUE if the message header is complete, otherwise FALSE.

Discussion
After calling `CFHTTPMessageAppendBytes` (page 54), call this function to see if the message header is complete.

Availability
Available in Mac OS X version 10.1 and later.

Declared In
`CFHTTPMessage.h`

CFHTTPMessageIsRequest

Returns a boolean indicating whether the CFHTTPMessage is a request or a response.

```
extern Boolean CFHTTPMessageIsRequest(CFHTTPMessageRef message);
```

Availability
Available in Mac OS X version 10.1 and later.

Declared In
`CFHTTPMessage.h`

CFHTTPMessageSetBody

Sets the body of a `CFHTTPMessage` object.

```
void CFHTTPMessageSetBody (
   CFHTTPMessageRef message,
   CFDataRef bodyData
);
```

Parameters

message
The message to modify.

bodyData
The data that is to be set as the body of the message.

Availability
Available in Mac OS X version 10.1 and later.

Declared In
`CFHTTPMessage.h`

CFHTTPMessageSetHeaderFieldValue

Sets the value of a header field in an HTTP message.

```
void CFHTTPMessageSetHeaderFieldValue (
   CFHTTPMessageRef message,
   CFStringRef headerField,
   CFStringRef value
);
```

Parameters

message
The message to modify.

headerField
The header field to set.

value
The value to set.

Availability
Available in Mac OS X version 10.1 and later.

Declared In
`CFHTTPMessage.h`

Data Types

CFHTTPMessageRef

An opaque reference representing an HTTP message.

```
typedef struct __CFHTTPMessage *CFHTTPMessageRef;
```

Availability
Available in Mac OS X version 10.1 and later.

Declared In
`CFHTTPMessage.h`

Constants

CFHTTP Version Constants

Sets the HTTP version in a `CFHTTPMessage` request or response object.

```
const CFStringRef kCFHTTPVersion1_0;
const CFStringRef kCFHTTPVersion1_1;
```

Constants
`kCFHTTPVersion1_0`
Specifies HTTP version 1.0.

Available in Mac OS X version 10.1 and later.

`kCFHTTPVersion1_1`
Specifies HTTP version 1.1.

Available in Mac OS X version 10.1 and later.

Discussion
The HTTP version constants are used when you call `CFHTTPMessageCreateRequest` (page 61) and `CFHTTPMessageCreateResponse` (page 62) to create a request or response message.

Declared In
`CFNetwork/CFHTTPMessage.h`

Authentication Schemes

Constants used to specify the desired authentication scheme for a request.

```
extern const CFStringRef kCFHTTPAuthenticationSchemeBasic;
extern const CFStringRef kCFHTTPAuthenticationSchemeDigest;
extern const CFStringRef kCFHTTPAuthenticationSchemeNTLM;
extern const CFStringRef kCFHTTPAuthenticationSchemeNegotiate;
```

Constants
`kCFHTTPAuthenticationSchemeBasic`
Request the HTTP basic authentication scheme.

Available in Mac OS X version 10.2 and later.

Declared in `CFHTTPMessage.h`.

`kCFHTTPAuthenticationSchemeDigest`
Request the HTTP digest authentication scheme.

Available in Mac OS X version 10.2 and later.

Declared in `CFHTTPMessage.h`.

`kCFHTTPAuthenticationSchemeNTLM`
Request the HTTP NTLM authentication scheme.

Available in Mac OS X version 10.5 and later.

Declared in `CFHTTPMessage.h`.

`kCFHTTPAuthenticationSchemeNegotiate`
Request an automatically negotiated authentication scheme.

Available in Mac OS X version 10.5 and later.

Declared in `CFHTTPMessage.h`.

CFNetDiagnostics Reference

Derived From:	CFType
Framework:	CoreServices
Declared in	CFNetwork/CFNetDiagnostics.h
Companion guide	CFNetwork Programming Guide

Overview

The CFNetDiagnostics opaque type allows you to diagnose network-related problems.

Functions by Task

Creating a net diagnostics object

`CFNetDiagnosticCreateWithStreams` (page 68)
Creates a network diagnostic object from a pair of CFStreams.

`CFNetDiagnosticCreateWithURL` (page 69)
Creates a CFNetDiagnosticRef from a CFURLRef.

CFNetDiagnostics Functions

`CFNetDiagnosticSetName` (page 71)
Overrides the displayed application name.

`CFNetDiagnosticDiagnoseProblemInteractively` (page 70)
Opens a Network Diagnostics window.

`CFNetDiagnosticCopyNetworkStatusPassively` (page 68)
Gets a network status value.

Functions

CFNetDiagnosticCopyNetworkStatusPassively

Gets a network status value.

```
CFNetDiagnosticStatus CFNetDiagnosticCopyNetworkStatusPassively (
   CFNetDiagnosticRef details,
   CFStringRef *description
);
```

Parameters

details

CFNetDiagnosticRef, created by `CFNetDiagnosticCreateWithStreams` (page 68) or `CFNetDiagnosticCreateWithURL` (page 69), for which the Network Diagnostics status is to be obtained.

description

If not `NULL`, upon return contains a localized string containing a description of the current network status. Ownership follows the Create Rule.

Return Value

A network status value.

Discussion

This function returns a status value that can be used to display basic information about the connection, and optionally gets a localized string containing a description of the current network status.

This function is guaranteed not to generate network activity.

Special Considerations

This function is thread safe as long as another thread does not alter the same CFNetDiagnosticRef at the same time.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CFNetDiagnostics.h`

CFNetDiagnosticCreateWithStreams

Creates a network diagnostic object from a pair of CFStreams.

```
CFNetDiagnosticRef CFNetDiagnosticCreateWithStreams (
   CFAllocatorRef alloc,
   CFReadStreamRef readStream,
   CFWriteStreamRef writeStream
);
```

Parameters

alloc

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

readStream

Reference to a read stream whose connection has failed, or `NULL` if you do not want the CFNetDiagnosticRef to have a read stream.

writeStream

Reference to a write stream whose connection has failed, or `NULL` if you do not want the CFNetDiagnosticRef to have a write stream.

function result

CFNetDiagnosticRef that you can pass to `CFNetDiagnosticDiagnoseProblemInteractively` (page 70) or `CFNetDiagnosticCopyNetworkStatusPassively` (page 68). Ownership follows the Create Rule.

Discussion

This function uses references to a read steam and a write stream (or just a read stream or just a write stream) to create a reference to an instance of a CFNetDiagnostic object. You can pass the reference to `CFNetDiagnosticDiagnoseProblemInteractively` (page 70) to open a Network Diagnostics window or to `CFNetDiagnosticCopyNetworkStatusPassively` (page 68) to get a description of the connection referenced by `readStream` and `writeStream`.

Special Considerations

This function is thread safe as long as another thread does not alter the same CFNetDiagnosticRef at the same time.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CFNetDiagnostics.h`

CFNetDiagnosticCreateWithURL

Creates a CFNetDiagnosticRef from a CFURLRef.

```
CFNetDiagnosticRef CFNetDiagnosticCreateWithURL (
   CFAllocatorRef alloc,
   CFURLRef url
);
```

Parameters

alloc

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

url

CFURLRef that refers to the failed connection.

Return Value
CFNetDiagnosticRef that you can pass to `CFNetDiagnosticDiagnoseProblemInteractively` (page 70) or `CFNetDiagnosticCopyNetworkStatusPassively` (page 68). Ownership follows the Create Rule.

Discussion
This function uses a URL to create a reference to an instance of a CFNetDiagnostic object. You can pass the reference to `CFNetDiagnosticDiagnoseProblemInteractively` (page 70) to open a Network Diagnostics window or to `CFNetDiagnosticCopyNetworkStatusPassively` (page 68) to get a description of the connection referenced by `readStream` and `writeStream`.

Special Considerations

This function is thread safe as long as another thread does not alter the same CFNetDiagnosticRef at the same time.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFNetDiagnostics.h`

CFNetDiagnosticDiagnoseProblemInteractively

Opens a Network Diagnostics window.

```
CFNetDiagnosticStatus CFNetDiagnosticDiagnoseProblemInteractively (
   CFNetDiagnosticRef details
);
```

Parameters

details

A network diagnostics object, created by `CFNetDiagnosticCreateWithStreams` (page 68) or `CFNetDiagnosticCreateWithURL` (page 69), for which the window is to be opened.

Return Value
`CFNetDiagnosticNoErr` if no error occurred, or `CFNetDiagnosticErr` if an error occurred that prevented this call from completing successfully.

Discussion
This function opens the Network Diagnostics window and returns immediately once the window is open.

Special Considerations

This function is thread safe as long as another thread does not alter the same CFNetDiagnosticRef at the same time.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFNetDiagnostics.h`

CFNetDiagnosticSetName

Overrides the displayed application name.

```
void CFNetDiagnosticSetName (
   CFNetDiagnosticRef details,
   CFStringRef name
);
```

Parameters

details

The network diagnostics object for which the application name is to be set.

name

Name that is to be set.

Discussion

Frameworks requiring that an application name be displayed to the user derive the application name from the bundle identifier of the currently running application, in that application's localization. If you want to override the derived application name, use this function to set the name that is displayed.

Special Considerations

This function is thread safe as long as another thread does not alter the same CFNetDiagnosticRef at the same time.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CFNetDiagnostics.h`

Data Types

CFNetDiagnosticRef

An opaque reference representing a CFNetDiagnostic.

```
typedef struct __CFNetDiagnostic* CFNetDiagnosticRef;
```

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CFNetDiagnostics.h`

CFNetDiagnosticStatus

A CFIndex type that is used to return status values from `CFNetDiagnostic` status and diagnostic functions. For a list of possible values, see "CFNetDiagnosticStatusValues Constants" (page 72).

```
typedef CFIndex                         CFNetDiagnosticStatus;
```

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFNetDiagnostics.h`

Constants

CFNetDiagnosticStatusValues Constants

Constants for diagnostic status values.

```
enum CFNetDiagnosticStatusValues {
    kCFNetDiagnosticNoErr = 0,
    kCFNetDiagnosticErr = -66560L,
    kCFNetDiagnosticConnectionUp = -66559L,
    kCFNetDiagnosticConnectionIndeterminate = -66558L,
    kCFNetDiagnosticConnectionDown = -66557L
};
typedef enum CFNetDiagnosticStatusValues CFNetDiagnosticStatusValues;
```

Constants

`kCFNetDiagnosticNoErr`
No error occurred but there is no status.
Available in Mac OS X v10.4 and later.
Declared in `CFNetDiagnostics.h`.

`kCFNetDiagnosticErr`
An error occurred that prevented the call from completing.
Available in Mac OS X v10.4 and later.
Declared in `CFNetDiagnostics.h`.

`kCFNetDiagnosticConnectionUp`
The connection appears to be working.
Available in Mac OS X v10.4 and later.
Declared in `CFNetDiagnostics.h`.

`kCFNetDiagnosticConnectionIndeterminate`
The status of the connection is not known.
Available in Mac OS X v10.4 and later.
Declared in `CFNetDiagnostics.h`.

`kCFNetDiagnosticConnectionDown`
The connection does not appear to be working.
Available in Mac OS X v10.4 and later.
Declared in `CFNetDiagnostics.h`.

Discussion
Diagnostic status values are returned by `CFNetDiagnosticDiagnoseProblemInteractively` (page 70) and `CFNetDiagnosticCopyNetworkStatusPassively` (page 68).

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFNetwork/CFNetDiagnostics.h`

CFNetServices Reference

Derived From:	CFType
Framework:	CoreServices
Declared in	CFNetwork/CFNetServices.h
Companion guides	Bonjour Overview CFNetwork Programming Guide NSNetServices and CFNetServices Programming Guide

Overview

The CFNetServices API is part of Bonjour, Apple's implementation of zero-configuration networking (ZEROCONF). The CFNetServices API allows you to register a network service, such as a printer or file server, so that it can be found by name or browsed for by service type and domain. Applications can use the CFNetServices API to discover the services that are available on the network and to find all access information — such as name, IP address, and port number — needed to use each service.

In effect, Bonjour registration and discovery combine the functions of a local DNS server and AppleTalk, allowing applications to provide the kind of user-friendly browsing available in the AppleTalk Chooser using open protocols, such as Multicast DNS (mDNS). Bonjour gives applications easy access to services over local IP networks without requiring the service to support an AppleTalk stack, and without requiring a DNS server on the local network.

For a full description of Bonjour, see *Bonjour Overview*.

Functions by Task

Creating net service objects

`CFNetServiceCreate` (page 84)
Creates an instance of a Network Service object.

`CFNetServiceCreateCopy` (page 86)
Creates a copy of a CFNetService object.

`CFNetServiceMonitorCreate` (page 92)
Creates an instance of a NetServiceMonitor object that watches for record changes.

`CFNetServiceBrowserCreate` (page 78)
Creates an instance of a Network Service browser object.

CFNetServices Functions

`CFNetServiceBrowserInvalidate` (page 79)
Invalidates an instance of a Network Service browser object.

`CFNetServiceBrowserScheduleWithRunLoop` (page 80)
Schedules a CFNetServiceBrowser on a run loop.

`CFNetServiceBrowserSearchForDomains` (page 80)
Searches for domains.

`CFNetServiceBrowserSearchForServices` (page 81)
Searches a domain for services of a specified type.

`CFNetServiceBrowserStopSearch` (page 82)
Stops a search for domains or services.

`CFNetServiceBrowserUnscheduleFromRunLoop` (page 83)
Unschedules a CFNetServiceBrowser from a run loop and mode.

`CFNetServiceCancel` (page 84)
Cancels a service registration or a service resolution.

`CFNetServiceCreateDictionaryWithTXTData` (page 86)
Uses TXT record data to create a dictionary.

`CFNetServiceCreateTXTDataWithDictionary` (page 87)
Flattens a set of key/value pairs into a CFDataRef suitable for passing to `CFNetServiceSetTXTData` (page 103).

`CFNetServiceGetAddressing` (page 88)
Gets the IP addressing from a CFNetService.

`CFNetServiceGetTargetHost` (page 90)
Queries a CFNetService for its target hosts.

`CFNetServiceGetDomain` (page 88)
Gets the domain from a CFNetService.

`CFNetServiceGetName` (page 89)
Gets the name from a CFNetService.

`CFNetServiceGetPortNumber` (page 89)
This function gets the port number from a CFNetService.

`CFNetServiceGetTXTData` (page 91)
Queries a network service for the contents of its TXT records.

`CFNetServiceGetType` (page 91)
Gets the type from a CFNetService.

`CFNetServiceMonitorInvalidate` (page 94)
Invalidates an instance of a Network Service monitor object.

`CFNetServiceMonitorScheduleWithRunLoop` (page 94)
Schedules a CFNetServiceMonitor on a run loop.

`CFNetServiceMonitorStart` (page 95)
Starts monitoring.

`CFNetServiceMonitorStop` (page 96)
Stops a CFNetServiceMonitor.

`CFNetServiceMonitorUnscheduleFromRunLoop` (page 97)
Unschedules a CFNetServiceMonitor from a run loop.

`CFNetServiceRegisterWithOptions` (page 98)
Makes a CFNetService available on the network.

`CFNetServiceResolveWithTimeout` (page 100)
Gets the IP address or addresses for a CFNetService.

`CFNetServiceScheduleWithRunLoop` (page 101)
Schedules a CFNetService on a run loop.

`CFNetServiceSetClient` (page 102)
Associates a callback function with a CFNetService or disassociates a callback function from a CFNetService.

`CFNetServiceSetTXTData` (page 103)
Sets the TXT record for a CFNetService.

`CFNetServiceUnscheduleFromRunLoop` (page 104)
Unschedules a CFNetService from a run loop.

`CFNetServiceGetProtocolSpecificInformation` (page 90) Deprecated in Mac OS X version 10.4
This function gets protocol-specific information from a CFNetService. (Deprecated. Use `CFNetServiceGetTXTData` (page 91) instead.)

`CFNetServiceRegister` (page 97) Deprecated in Mac OS X version 10.4
Makes a CFNetService available on the network. (Deprecated. Use `CFNetServiceRegisterWithOptions` (page 98) instead.)

`CFNetServiceResolve` (page 99) Deprecated in Mac OS X version 10.4
This function updates the specified CFNetService with the IP address or addresses associated with the service. Call `CFNetServiceGetAddressing` (page 88) to get the addresses. (Deprecated. Use `CFNetServiceResolveWithTimeout` (page 100) instead.)

Modifying a net service

`CFNetServiceSetProtocolSpecificInformation` (page 103) Deprecated in Mac OS X version 10.4
Sets protocol-specific information for a CFNetService. (Deprecated. Use `CFNetServiceSetTXTData` instead.)

Getting the net service type IDs

`CFNetServiceGetTypeID` (page 92)
Gets the Core Foundation type identifier for the Network Service object.

`CFNetServiceMonitorGetTypeID` (page 94)
Gets the Core Foundation type identifier for all CFNetServiceMonitor instances.

`CFNetServiceBrowserGetTypeID` (page 79)
Gets the Core Foundation type identifier for the Network Service browser object.

Functions

CFNetServiceBrowserCreate

Creates an instance of a Network Service browser object.

```
CFNetServiceBrowserRef CFNetServiceBrowserCreate (
   CFAllocatorRef alloc,
   CFNetServiceBrowserClientCallBack clientCB,
   CFNetServiceClientContext *clientContext
);
```

Parameters

alloc

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

clientCB

Callback function that is to be called when domains and services are found; cannot be `NULL`. For details, see `CFNetServiceBrowserClientCallBack` (page 104).

clientContext

Context information to be used when `clientCB` is called; cannot be `NULL`. For details, see `CFNetServiceClientContext` (page 107).

Return Value

A new browser object, or `NULL` if the instance could not be created. Ownership follows the Create Rule.

Discussion

This function creates an instance of a Network Service browser object, called a CFNetServiceBrowser, that can be used to search for domains and for services.

To use the resulting CFNetServiceBrowser in asynchronous mode, call `CFNetServiceBrowserScheduleWithRunLoop` (page 80). Then call `CFNetServiceBrowserSearchForDomains` (page 80) and `CFNetServiceBrowserSearchForServices` (page 81) to use the CFNetServiceBrowser to search for services and domains, respectively. The callback function specified by `clientCB` is called from a run loop to pass search results to your application. The search continues until you stop the search by calling `CFNetServiceBrowserStopSearch` (page 82).

If you do not call `CFNetServiceBrowserScheduleWithRunLoop` (page 80), searches with the resulting CFNetServiceBrowser are made in synchronous mode. Calls made to `CFNetServiceBrowserSearchForDomains` (page 80) and `CFNetServiceBrowserSearchForServices` (page 81) block until there are search results, in which case the callback function specified by `clientCB` is called, until the search is are stopped by calling `CFNetServiceBrowserStopSearch` (page 82) from another thread, or an error occurs.

To shut down a CFNetServiceBrowser that is running in asynchronous mode, call `CFNetServiceBrowserUnscheduleFromRunLoop` (page 83), followed by `CFNetServiceBrowserInvalidate` (page 79), and then `CFNetServiceBrowserStopSearch` (page 82).

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetServices.h`

CFNetServiceBrowserGetTypeID

Gets the Core Foundation type identifier for the Network Service browser object.

```
CFTypeID CFNetServiceBrowserGetTypeID ();
```

Return Value
The type ID.

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetServices.h`

CFNetServiceBrowserInvalidate

Invalidates an instance of a Network Service browser object.

```
void CFNetServiceBrowserInvalidate (
   CFNetServiceBrowserRef browser
);
```

Parameters

browser

The CFNetServiceBrowser to invalidate, obtained by a previous call to `CFNetServiceBrowserCreate` (page 78).

Discussion
This function invalidates the specified instance of a Network Service browser object. Any searches using the specified instance that are in progress when this function is called are stopped. An invalidated browser cannot be scheduled on a run loop and its callback function is never called.

Special Considerations

This function is thread safe as long as another thread does not alter the same CFNetServiceBrowserRef at the same time.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetServices.h`

CFNetServiceBrowserScheduleWithRunLoop

Schedules a CFNetServiceBrowser on a run loop.

```
void CFNetServiceBrowserScheduleWithRunLoop (
   CFNetServiceBrowserRef browser,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

browser

The CFNetServiceBrowser that is to be scheduled on a run loop; cannot be `NULL`.

runLoop

The run loop on which the browser is to be scheduled; cannot be `NULL`.

runLoopMode

The mode on which to schedule the browser; cannot be `NULL`.

Discussion

This function schedules the specified CFNetServiceBrowser on the run loop, thereby placing the browser in asynchronous mode. The run loop will call the browser's callback function to deliver the results of domain and service searches. The caller is responsible for ensuring that at least one of the run loops on which the browser is scheduled is being run.

Special Considerations

This function is thread safe.

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CFNetServices.h`

CFNetServiceBrowserSearchForDomains

Searches for domains.

```
Boolean CFNetServiceBrowserSearchForDomains (
   CFNetServiceBrowserRef browser,
   Boolean registrationDomains,
   CFStreamError *error
);
```

Parameters

browser

The CFNetServiceBrowser, obtained by previously calling `CFNetServiceBrowserCreate` (page 78), that is to perform the search; cannot be `NULL`.

registrationDomains

`TRUE` to search for only registration domains; `FALSE` to search for domains that can be browsed for services. For this version of the CFNetServices API, the registration domain is the local domain maintained by the mDNS responder running on the same machine as the calling application.

error

A pointer to a `CFStreamError` structure, that, if an error occurs, will be set to the error and the error's domain and passed to your callback function. Pass `NULL` if you don't want to receive the error that may occur as a result of this particular call.

Return Value

`TRUE` if the search was started (asynchronous mode); `FALSE` if another search is already in progress for this CFNetServiceBrowser or if an error occurred.

Discussion

This function uses a CFNetServiceBrowser to search for domains. The search continues until the search is canceled by calling `CFNetServiceBrowserStopSearch` (page 82). If `registrationDomains` is `TRUE`, this function searches only for domains in which services can be registered. If `registrationDomains` is `FALSE`, this function searches for domains that can be browsed for services. When a domain is found, the callback function specified when the CFNetServiceBrowser was created is called and passed an instance of a CFStringRef containing the domain that was found.

In asynchronous mode, this function returns `TRUE` if the search was started. Otherwise, it returns `FALSE`.

In synchronous mode, this function blocks until the search is stopped by calling `CFNetServiceBrowserStopSearch` (page 82) from another thread, in which case it returns `FALSE`, or until an error occurs.

Special Considerations

This function is thread safe.

For any one CFNetServiceBrowser, only one domain search or one service search can be in progress at the same time.

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CFNetServices.h`

CFNetServiceBrowserSearchForServices

Searches a domain for services of a specified type.

```
Boolean CFNetServiceBrowserSearchForServices (
   CFNetServiceBrowserRef browser,
   CFStringRef domain,
   CFStringRef serviceType,
   CFStreamError *error
);
```

Parameters

browser

The CFNetServiceBrowser, obtained by previously calling `CFNetServiceBrowserCreate` (page 78), that is to perform the search; cannot be `NULL`.

domain

The domain to search for the service type; cannot be `NULL`. To get the domains that are available for searching, call `CFNetServiceBrowserSearchForDomains` (page 80).

type

The service type to search for; cannot be `NULL`. For a list of valid service types, see http://www.iana.org/assignments/port-numbers.

error

A pointer to a `CFStreamError` structure, that, if an error occurs, will be set to the error and the error's domain and passed to your callback function. Pass `NULL` if you don't want to receive the error that may occur as a result of this particular call.

Return Value

`TRUE` if the search was started (asynchronous mode); `FALSE` if another search is already in progress for this CFNetServiceBrowser or if an error occurred.

Discussion

This function searches the specified domain for services that match the specified service type. The search continues until the search is canceled by calling `CFNetServiceBrowserStopSearch` (page 82). When a match is found, the callback function specified when the CFNetServiceBrowser was created is called and passed an instance of a CFNetService representing the service that was found.

In asynchronous mode, this function returns `TRUE` if the search was started. Otherwise, it returns `FALSE`.

In synchronous mode, this function blocks until the search is stopped by calling `CFNetServiceBrowserStopSearch` (page 82) from another thread, in which case this function returns `FALSE`, or until an error occurs.

Special Considerations

This function is thread safe.

For any one CFNetServiceBrowser, only one domain search or one service search can be in progress at the same time.

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CFNetServices.h`

CFNetServiceBrowserStopSearch

Stops a search for domains or services.

```
void CFNetServiceBrowserStopSearch (
   CFNetServiceBrowserRef browser,
   CFStreamError *error
);
```

Parameters

browser

The CFNetServiceBrowser that was used to start the search; cannot be `NULL`.

error

A pointer to a `CFStreamError` structure that will be passed to the callback function associated with this CFNetServiceBrowser (if the search is being conducted in asynchronous mode) or that is pointed to by the `error` parameter when `CFNetServiceBrowserSearchForDomains` (page 80) or `CFNetServiceBrowserSearchForServices` (page 81) returns (if the search is being conducted in synchronous mode). Set the `domain` field to `kCFStreamErrorDomainCustom` and the `error` field to an appropriate value.

Discussion

This functions stops a search started by a previous call to `CFNetServiceBrowserSearchForDomains` (page 80) or `CFNetServiceBrowserSearchForServices` (page 81). For asynchronous and synchronous searches, calling this function causes the callback function associated with the CFNetServiceBrowser to be called once for each domain or service found. If the search is asynchronous, `error` is passed to the callback function. If the search is synchronous, calling this function causes `CFNetServiceBrowserSearchForDomains` or `CFNetServiceBrowserSearchForServices` to return `FALSE`. If the `error` parameter for either call pointed to a `CFStreamError` structure, the `CFStreamError` structure contains the error code and the error code's domain as set when this function was called.

Special Considerations

This function is thread safe.

If you are stopping an asynchronous search, before calling this function, call `CFNetServiceBrowserUnscheduleFromRunLoop` (page 83), followed by `CFNetServiceBrowserInvalidate` (page 79).

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CFNetServices.h`

CFNetServiceBrowserUnscheduleFromRunLoop

Unschedules a CFNetServiceBrowser from a run loop and mode.

```
void CFNetServiceBrowserUnscheduleFromRunLoop (
   CFNetServiceBrowserRef browser,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

browser

The CFNetServiceBrowser that is to be unscheduled; cannot be `NULL`.

runLoop

The run loop; cannot be `NULL`.

runLoopMode

The mode from which the browser is to be unscheduled; cannot be `NULL`.

Discussion

Call this function to shut down a browser that is running asynchronously. To complete the shutdown, call `CFNetServiceBrowserInvalidate` (page 79) followed by `CFNetServiceBrowserStopSearch` (page 82).

Special Considerations

This function is thread safe.

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CFNetServices.h`

CFNetServiceCancel

Cancels a service registration or a service resolution.

```
void CFNetServiceCancel (
   CFNetServiceRef theService
);
```

Parameters

theService

The CFNetService, obtained by previously calling `CFNetServiceCreate` (page 84), for which a registration or a resolution is to be canceled.

Discussion

This function cancels service registrations, started by `CFNetServiceRegister` (page 97), thereby making the service unavailable. It also cancels service resolutions, started by `CFNetServiceResolve` (page 99).

If you are shutting down an asynchronous service, you should first call `CFNetServiceUnscheduleFromRunLoop` (page 104) and `CFNetServiceSetClient` (page 102) with `clientCB` set to `NULL`. Then call this function.

If you are shutting down a synchronous service, call this function from another thread.

This function also cancels service resolutions. You would want to cancel a service resolution if your callback function has received an IP address that you've successfully used to connect to the service. In addition, you might want to cancel a service resolution if the resolution is taking longer than a user would want to wait or if the user canceled the operation.

Special Considerations

This function is thread safe.

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CFNetServices.h`

CFNetServiceCreate

Creates an instance of a Network Service object.

```
CFNetServiceRef CFNetServiceCreate (
   CFAllocatorRef alloc,
   CFStringRef domain,
   CFStringRef serviceType,
   CFStringRef name,
   SInt32 port
);
```

Parameters

alloc

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

domain

The domain in which the CFNetService is to be registered; cannot be `NULL`. Call `CFNetServiceBrowserCreate` (page 78) and `CFNetServiceBrowserSearchForDomains` (page 80) to get the registration domain.

type

The type of service being registered; cannot be `NULL`. For a list of valid service types, see http://www.iana.org/assignments/port-numbers.

name

A unique name if the instance will be used to register a service. The name will become part of the instance name in the DNS records that will be created when the service is registered. If the instance will be used to resolve a service, the name should be the name of the machine or service that will be resolved.

port

Local IP port, in host byte order, on which this service accepts connections. Pass zero to get placeholder service. With a placeholder service, the service will not be discovered by browsing, but a name conflict will occur if another client tries to register the same name. Most applications do not need to use placeholder service.

Return Value

A new net service object, or `NULL` if the instance could not be created. Ownership follows the Create Rule.

Discussion

If the service depends on information in DNS TXT records, call `CFNetServiceSetProtocolSpecificInformation` (page 103).

If the CFNetService is to run in asynchronous mode, call `CFNetServiceSetClient` (page 102) to prepare the service for running in asynchronous mode. Then call `CFNetServiceScheduleWithRunLoop` (page 101) to schedule the service on a run loop. Then call `CFNetServiceRegister` (page 97) to make the service available.

If the CFNetService is to run in synchronous mode, call `CFNetServiceRegister` (page 97).

To terminate a service that is running in asynchronous mode, call `CFNetServiceCancel` (page 84) and `CFNetServiceUnscheduleFromRunLoop` (page 104).

To terminate a service that is running in synchronous mode, call `CFNetServiceCancel` (page 84).

Special Considerations

This function is thread safe.

Availability

Available in Mac OS X version 10.2 and later.

Declared In
CFNetServices.h

CFNetServiceCreateCopy

Creates a copy of a CFNetService object.

```
CFNetServiceRef CFNetServiceCreateCopy (
   CFAllocatorRef alloc,
   CFNetServiceRef service
);
```

Parameters

alloc

The allocator to use to allocate memory for the new object. Pass NULL or kCFAllocatorDefault to use the current default allocator.

service

CFNetServiceRef to be copied; cannot be NULL. If service is not a valid CFNetServiceRef, the behavior of this function is undefined.

Return Value
Copy of service, including all previously resolved data, or NULL if service could not be copied. Ownership follows the Create Rule.

Discussion
This function creates a copy of the CFNetService specified by service.

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
CFNetServices.h

CFNetServiceCreateDictionaryWithTXTData

Uses TXT record data to create a dictionary.

```
CFDictionaryRef CFNetServiceCreateDictionaryWithTXTData (
   CFAllocatorRef alloc,
   CFDataRef txtRecord
);
```

Parameters

alloc

The allocator to use to allocate memory for the new object. Pass NULL or kCFAllocatorDefault to use the current default allocator.

txtRecord

TXT record data as returned by CFNetServiceGetTXTData (page 91).

Return Value
A dictionary containing the key/value pairs parsed from `txtRecord`, or `NULL` if `txtRecord` cannot be parsed. Each key in the dictionary is a CFString object, and each value is a CFData object. Ownership follows the Create Rule.

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFNetServices.h`

CFNetServiceCreateTXTDataWithDictionary

Flattens a set of key/value pairs into a CFDataRef suitable for passing to `CFNetServiceSetTXTData` (page 103).

```
CFDataRef CFNetServiceCreateTXTDataWithDictionary (
   CFAllocatorRef alloc,
   CFDictionaryRef keyValuePairs
);
```

Parameters

alloc

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

keyValuePairs

CFDictionaryRef containing the key/value pairs that are to be placed in a TXT record. Each key must be a CFStringRef and each value should be a CFDataRef or a CFStringRef. (See the discussion below for additional information about values that are CFStringRefs.) This function fails if any other data types are provided. The length of a key and its value should not exceed 255 bytes.

Return Value
A CFData object containing the flattened form of *keyValuePairs*, or `NULL` if the dictionary could not be flattened. Ownership follows the Create Rule.

Discussion
This function flattens the key/value pairs in the dictionary specified by `keyValuePairs` into a CFDataRef suitable for passing to `CFNetServiceSetTXTData` (page 103). Note that this function is not a general purpose function for flattening CFDictionaryRefs.

The keys in the dictionary referenced by `keyValuePairs` must be CFStringRefs and the values must be CFDataRefs. Any values that are CFStringRefs are converted to CFDataRefs representing the flattened UTF-8 bytes of the string. The types of the values are not encoded in the CFDataRefs, so any CFStringRefs that are converted to CFDataRefs remain CFDataRefs when the CFDataRef produced by this function is processed by `CFNetServiceCreateDictionaryWithTXTData` (page 86).

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
CFNetServices.h

CFNetServiceGetAddressing

Gets the IP addressing from a CFNetService.

```
CFArrayRef CFNetServiceGetAddressing (
   CFNetServiceRef theService
);
```

Parameters

theService
The CFNetService whose IP addressing is to be obtained; cannot be NULL.

Return Value
A CFArray containing a CFDataRef for each IP address returned, or NULL. Each CFDataRef consists of a sockaddr structure containing the IP address of the service. This function returns NULL if the service's addressing is unknown because CFNetServiceResolve (page 99) has not been called for theService.

Discussion
This function gets the IP addressing from a CFNetService. Typically, the CFNetService was obtained by calling CFNetServiceBrowserSearchForServices (page 81). Before calling this function, call CFNetServiceResolve (page 99) to update the CFNetService with its IP addressing.

Special Considerations

This function gets the data in a thread-safe way, but the data itself is not safe if the service is altered from another thread.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
CFNetServices.h

CFNetServiceGetDomain

Gets the domain from a CFNetService.

```
CFStringRef CFNetServiceGetDomain (
   CFNetServiceRef theService
);
```

Parameters

theService
The CFNetService whose domain is to be obtained; cannot be NULL.

Return Value
A CFString object containing the domain of the CFNetService.

Discussion
This function gets the domain from a CFNetService.

Special Considerations

This function is thread safe. The function gets the data in a thread-safe way, but the data is not safe if the service is altered from another thread.

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CFNetServices.h`

CFNetServiceGetName

Gets the name from a CFNetService.

```
CFStringRef CFNetServiceGetName (
   CFNetServiceRef theService
);
```

Parameters

`theService`

The CFNetService whose name is to be obtained; cannot be `NULL`.

Return Value

A CFString object containing the name of the service represented by the CFNetService.

Discussion

This function gets the name from a CFNetService.

Special Considerations

This function is thread safe. The function gets the data in a thread-safe way, but the data is not safe if the service is altered from another thread.

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CFNetServices.h`

CFNetServiceGetPortNumber

This function gets the port number from a CFNetService. (Deprecated in Mac OS X version 10.4.)

```
extern SInt32 CFNetServiceGetPortNumber(
    CFNetServiceRef theService);
```

Parameters

`theService`

The CFNetService whose protocol-specific information is to be obtained; cannot be `NULL`. Note that in order to get protocol-specific information, you must resolve `theService` by calling `CFNetServiceResolve` (page 99) or `CFNetServiceResolveWithTimeout` (page 100) before calling this function.

Return Value

A CFString object containing the protocol-specific information, or `NULL` if there is no information.

Special Considerations

This function gets the data in a thread-safe way, but the data itself is not safe if the service is altered from another thread.

Availability

Available in Mac OS X version 10.2 and later.

Deprecated in Mac OS X version 10.4.

Declared In

`CFNetServices.h`

CFNetServiceGetProtocolSpecificInformation

This function gets protocol-specific information from a CFNetService. (Deprecated in Mac OS X version 10.4. Use `CFNetServiceGetTXTData` (page 91) instead.)

```
CFStringRef CFNetServiceGetProtocolSpecificInformation (
   CFNetServiceRef theService
);
```

Parameters

`theService`

The CFNetService whose protocol-specific information is to be obtained; cannot be `NULL`. Note that in order to get protocol-specific information, you must resolve `theService` by calling `CFNetServiceResolve` (page 99) or `CFNetServiceResolveWithTimeout` (page 100) before calling this function.

Return Value

A CFString object containing the protocol-specific information, or `NULL` if there is no information.

Special Considerations

This function gets the data in a thread-safe way, but the data itself is not safe if the service is altered from another thread.

Availability

Available in Mac OS X version 10.2 and later.

Deprecated in Mac OS X version 10.4.

Declared In

`CFNetServices.h`

CFNetServiceGetTargetHost

Queries a CFNetService for its target hosts.

```
CFStringRef CFNetServiceGetTargetHost (
   CFNetServiceRef theService
);
```

Parameters

`theService`

Network service to be queried.

Return Value
The target host name of the machine providing the service or `NULL` is of the service's target host is not known. (The target host will not be known if it has not been resolved.)

Special Considerations

This function is thread safe, but the target host name is not safe if the service is altered from another thread.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFNetServices.h`

CFNetServiceGetTXTData

Queries a network service for the contents of its TXT records.

```
CFDataRef CFNetServiceGetTXTData (
   CFNetServiceRef theService
);
```

Parameters

`theService`

Reference for the network service whose TXT record data is to be obtained; cannot be `NULL`. Note that in order to get TXT record data, you must resolve `theService` by calling `CFNetServiceResolve` (page 99) or `CFNetServiceResolveWithTimeout` (page 100) before calling this function.

Return Value
CFDataRef object containing the requested TXT data and suitable for passing to `CFNetServiceCreateDictionaryWithTXTData` (page 86), or `NULL` if the service's TXT data has not been resolved.

Discussion
This function gets the data from the service's TXT records.

Special Considerations

This function gets the data in a thread-safe way, but the data itself is not safe if the service is altered from another thread.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFNetServices.h`

CFNetServiceGetType

Gets the type from a CFNetService.

```
CFStringRef CFNetServiceGetType (
   CFNetServiceRef theService
);
```

Parameters

theService

The CFNetService whose type is to be obtained; cannot be `NULL`.

Return Value
A CFString object containing the type from a CFNetService.

Discussion
This function gets the type of a CFNetService.

Special Considerations

This function is thread safe. The function gets the data in a thread-safe way, but the data is not safe if the service is altered from another thread.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetServices.h`

CFNetServiceGetTypeID

Gets the Core Foundation type identifier for the Network Service object.

```
CFTypeID CFNetServiceGetTypeID ();
```

Return Value
The type ID.

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetServices.h`

CFNetServiceMonitorCreate

Creates an instance of a NetServiceMonitor object that watches for record changes.

```
CFNetServiceMonitorRef CFNetServiceMonitorCreate (
   CFAllocatorRef alloc,
   CFNetServiceRef theService,
   CFNetServiceMonitorClientCallBack clientCB,
   CFNetServiceClientContext *clientContext
);
```

Parameters

alloc

The allocator to use to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

theService

CFNetService to be monitored.

clientCB

Pointer to callback function that is to be called when a record associated with `theService` changes; cannot be `NULL`.

clientContext

Pointer to user-defined contextual information that is to be passed to the callback specified by `clientCB` when the callback is called; cannot be `NULL`. For details, see CFNetServiceClientContext (page 107).

Return Value

A new instance of a CFNetServiceMonitor, or `NULL` if the monitor could not be created. Ownership follows the Create Rule.

Discussion

This function creates a CFNetServiceMonitor that watches for changes in records associated with `theService`.

If the CFNetServiceMonitor is to run in asynchronous mode, call `CFNetServiceMonitorScheduleWithRunLoop` (page 94) to schedule the monitor on a run loop. Then call `CFNetServiceMonitorStart` (page 95) to start monitoring. When a change occurs, the callback function specified by `clientCB` will be called. For details, see CFNetServiceMonitorClientCallBack (page 106).

If the CFNetServiceMonitor is to run in synchronous mode, call `CFNetServiceMonitorStart` (page 95).

To stop a monitor that is running in asynchronous mode, call `CFNetServiceMonitorStop` (page 96) and `CFNetServiceMonitorUnscheduleFromRunLoop` (page 97).

To stop a monitor that is running in synchronous mode, call `CFNetServiceMonitorStop` (page 96).

If you no longer need to monitor record changes, call `CFNetServiceMonitorStop` (page 96) to stop the monitor and then call `CFNetServiceMonitorInvalidate` (page 94)to invalidate the monitor so it cannot be used again. Then call `CFRelease` to release the memory associated with CFNetServiceMonitorRef.

Special Considerations

This function is thread safe.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CFNetServices.h`

CFNetServiceMonitorGetTypeID

Gets the Core Foundation type identifier for all CFNetServiceMonitor instances.

```
CFTypeID CFNetServiceMonitorGetTypeID ();
```

Return Value
The type ID.

Special Considerations

This function is thread safe.

Version Notes
Introduced in Mac OS X v10.4.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetServices.h`

CFNetServiceMonitorInvalidate

Invalidates an instance of a Network Service monitor object.

```
void CFNetServiceMonitorInvalidate (
   CFNetServiceMonitorRef monitor
);
```

Parameters

monitor
CFNetServiceMonitor to invalidate; cannot be `NULL`.

Discussion
This function invalidates the specified Network Service monitor so that it cannot be used again. Before you call this function, you should call `CFNetServiceMonitorStop` (page 96). If the monitor has not already been stopped, this function stops the monitor for you.

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFNetServices.h`

CFNetServiceMonitorScheduleWithRunLoop

Schedules a CFNetServiceMonitor on a run loop.

```
void CFNetServiceMonitorScheduleWithRunLoop (
   CFNetServiceMonitorRef monitor,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

theService

The CFNetServiceMonitor that is to be scheduled on a run loop; cannot be `NULL`.

runLoop

The run loop on which the monitor is to be scheduled; cannot be `NULL`.

runLoopMode

The mode on which to schedule the monitor; cannot be `NULL`.

Discussion

Schedules the specified monitor on a run loop, which places the monitor in asynchronous mode. The caller is responsible for ensuring that at least one of the run loops on which the monitor is scheduled is being run.

Special Considerations

This function is thread safe.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CFNetServices.h`

CFNetServiceMonitorStart

Starts monitoring.

```
Boolean CFNetServiceMonitorStart (
   CFNetServiceMonitorRef monitor,
   CFNetServiceMonitorType recordType,
   CFStreamError *error
);
```

Parameters

monitor

CFNetServiceMonitor, created by calling `CFNetServiceMonitorCreate` (page 92), that is to be started.

recordType

CFNetServiceMonitorType that specified the type of record to monitor. For possible values, see CFNetServiceMonitorType Constants (page 110).

error

Pointer to a `CFStreamError` structure. If an error occurs, on output, the structure's `domain` field will be set to the error code's domain and the `error` field will be set to an appropriate error code. Set this parameter to `NULL` if you don't want to receive the error code and its domain.

Return Value

`TRUE` if an asynchronous monitor was started successfully. `FALSE` if an error occurred when starting an asynchronous or synchronous monitor, or if `CFNetServiceMonitorStop` (page 96) was called for an synchronous monitor.

Discussion
This function starts monitoring for changes to records of the type specified by `recordType`. If a monitor is already running for the service associated with the specified CFNetServiceMonitorRef, this function returns `FALSE`.

For synchronous monitors, this function blocks until the monitor is stopped by calling `CFNetServiceMonitorStop` (page 96), in which case, this function returns `FALSE`.

For asynchronous monitors, this function returns `TRUE` or `FALSE`, depending on whether monitoring starts successfully.

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFNetServices.h`

CFNetServiceMonitorStop

Stops a CFNetServiceMonitor.

```
void CFNetServiceMonitorStop (
   CFNetServiceMonitorRef monitor,
   CFStreamError *error
);
```

Parameters

monitor

CFNetServiceMonitor, started by calling `CFNetServiceMonitorStart` (page 95), that is to be stopped.

error

Pointer to a `CFStreamError` structure or `NULL`. For synchronous monitors, set the `error` field of this structure to the non-zero value you want to be set in the `CFStreamError` structure when `CFNetServiceMonitorStart` (page 95) returns. Note that when it returns, `CFNetServiceMonitorStart` returns `FALSE`. If the monitor was started asynchronously, set the `error` field to the non-zero value you want the monitor's callback to receive when it is called. If this parameter is `NULL`, default values for the `CFStreamError` structure are used: the domain is set to `kCFStreamErrorDomainNetServices` and the error code is set to `kCFNetServicesErrorCancel`.

Discussion
This function stops the specified monitor. Call `CFNetServiceMonitorStart` (page 95) if you want to start monitoring again.

If you want to stop monitoring and no longer need to monitor record changes, call `CFNetServiceMonitorInvalidate` (page 94) instead of this function.

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
CFNetServices.h

CFNetServiceMonitorUnscheduleFromRunLoop

Unschedules a CFNetServiceMonitor from a run loop.

```
void CFNetServiceMonitorUnscheduleFromRunLoop (
   CFNetServiceMonitorRef monitor,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

monitor
The CFNetServiceMonitor that is to be unscheduled; cannot be NULL.

runLoop
The run loop; cannot be NULL.

runLoopMode
The mode from which the monitor is to be unscheduled; cannot be NULL.

Discussion
Unschedules the specified monitor from the specified run loop and mode. Call this function to shut down a monitor that is running asynchronously.

To change a monitor so that it cannot be scheduled and so that its callback will never be called, call CFNetServiceMonitorInvalidate (page 94).

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
CFNetServices.h

CFNetServiceRegister

Makes a CFNetService available on the network. (Deprecated in Mac OS X version 10.4. Use CFNetServiceRegisterWithOptions (page 98) instead.)

```
Boolean CFNetServiceRegister (
   CFNetServiceRef theService,
   CFStreamError *error
);
```

Parameters

theService
The CFNetService to register; cannot be NULL. The registration will fail if the service doesn't have a domain, a type, a name, and an IP address.

error

A pointer to a `CFStreamError` structure that will be set to an error code and the error code's domain if an error occurs; or `NULL` if you don't want to receive the error code and its domain.

Return Value

`TRUE` if an asynchronous service registration was started; `FALSE` if an asynchronous or synchronous registration failed or if a synchronous registration was canceled.

Discussion

If the service is to run in asynchronous mode, you must call `CFNetServiceSetClient` (page 102) to associate a callback function with this CFNetService before calling this function.

When registering a service that runs in asynchronous mode, this function returns `TRUE` if the service contains all of the required attributes and the registration process can start. If the registration process completes successfully, the service is available on the network until you shut down the service by calling `CFNetServiceUnscheduleFromRunLoop` (page 104), `CFNetServiceSetClient` (page 102), and `CFNetServiceCancel` (page 84). If the service does not contain all of the required attributes or if the registration process does not complete successfully, this function returns `FALSE`.

When registering a service that runs in synchronous mode, this function blocks until an error occurs, in which case this function returns `FALSE`. Until this function returns `FALSE`, the service is available on the network. To force this function to return `FALSE`, thereby shutting down the service, call `CFNetServiceCancel` (page 84) from another thread.

Special Considerations

This function is thread safe.

Availability

Available in Mac OS X version 10.2 and later.

Deprecated in Mac OS X version 10.4.

Declared In

`CFNetServices.h`

CFNetServiceRegisterWithOptions

Makes a CFNetService available on the network.

```
Boolean CFNetServiceRegisterWithOptions (
   CFNetServiceRef theService,
   CFOptionFlags options,
   CFStreamError *error
);
```

Parameters

theService

Network service to register; cannot be `NULL`. The registration will fail if the service doesn't have a domain, a type, a name, and an IP address.

options

Bit flags for specifying registration options. Currently, the only registration option is `kCFNetServiceFlagNoAutoRename`. For details, see CFNetService Registration Options (page 109).

error

Pointer to a `CFStreamError` structure that will be set to an error code and the error code's domain if an error occurs; or `NULL` if you don't want to receive the error code and its domain.

Return Value

`TRUE` if an asynchronous service registration was started; `FALSE` if an asynchronous or synchronous registration failed or if a synchronous registration was canceled.

Discussion

If the service is to run in asynchronous mode, you must call `CFNetServiceSetClient` (page 102) to associate a callback function with this CFNetService before calling this function.

When registering a service that runs in asynchronous mode, this function returns `TRUE` if the service contains all of the required attributes and the registration process can start. If the registration process completes successfully, the service is available on the network until you shut down the service by calling `CFNetServiceUnscheduleFromRunLoop` (page 104), `CFNetServiceSetClient` (page 102), and `CFNetServiceCancel` (page 84). If the service does not contain all of the required attributes or if the registration process does not complete successfully, this function returns `FALSE`.

When registering a service that runs in synchronous mode, this function blocks until an error occurs, in which case this function returns `FALSE`. Until this function returns `FALSE`, the service is available on the network. To force this function to return `FALSE`, thereby shutting down the service, call `CFNetServiceCancel` (page 84) from another thread.

The `options` parameter is a bit flag for specifying service registration options. Currently, `kCFNetServiceFlagNoAutoRename` is the only supported registration option. If this bit is set and a service of the same name is running, the registration will fail. If this bit is not set and a service of the same name is running, the service that is being registered will be renamed automatically by appending `(n)` to the service name, where *n* is a number that is incremented until the service can be registered with a unique name.

Special Considerations

This function is thread safe.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`CFNetServices.h`

CFNetServiceResolve

This function updates the specified CFNetService with the IP address or addresses associated with the service. Call `CFNetServiceGetAddressing` (page 88) to get the addresses. (Deprecated in Mac OS X version 10.4. Use `CFNetServiceResolveWithTimeout` (page 100) instead.)

```
Boolean CFNetServiceResolve (
   CFNetServiceRef theService,
   CFStreamError *error
);
```

Parameters

theService

The CFNetService to resolve; cannot be `NULL`. The resolution will fail if the service doesn't have a domain, a type, and a name.

error

A pointer to a `CFStreamError` structure that will be set to an error code and the error code's domain if an error occurs; or `NULL` if you don't want to receive the error code and its domain.

Return Value

`TRUE` if an asynchronous service resolution was started or if a synchronous service resolution updated the CFNetService; `FALSE` if an asynchronous or synchronous resolution failed or if a synchronous resolution was canceled.

Discussion

When resolving a service that runs in asynchronous mode, this function returns `TRUE` if the CFNetService has a domain, type, and name, and the underlying resolution process was started. Otherwise, this function returns `FALSE`. Once started, the resolution continues until it is canceled by calling `CFNetServiceCancel` (page 84).

When resolving a service that runs in synchronous mode, this function blocks until the CFNetService is updated with at least one IP address, until an error occurs, or until `CFNetServiceCancel` (page 84) is called.

Special Considerations

This function is thread safe.

If the service will be used in asynchronous mode, you must call `CFNetServiceSetClient` (page 102) before calling this function.

Availability

Available in Mac OS X version 10.2 and later.

Deprecated in Mac OS X version 10.4.

Declared In

`CFNetServices.h`

CFNetServiceResolveWithTimeout

Gets the IP address or addresses for a CFNetService.

```
Boolean CFNetServiceResolveWithTimeout (
   CFNetServiceRef theService,
   CFTimeInterval timeout,
   CFStreamError *error
);
```

Parameters

theService

The CFNetService to resolve; cannot be `NULL`. The resolution will fail if the service doesn't have a domain, a type, and a name.

timeout

Value of type `CFTimeInterval` specifying the maximum amount of time allowed to perform the resolution. If the resolution is not performed within the specified amount of time, a timeout error will be returned. If `timeout` is less than or equal to zero, an infinite amount of time is allowed.

error

Pointer to a `CFStreamError` structure that will be set to an error code and the error code's domain if an error occurs; or `NULL` if you don't want to receive the error code and its domain.

Return Value
`TRUE` if an asynchronous service resolution was started or if a synchronous service resolution updated the CFNetService; `FALSE` if an asynchronous or synchronous resolution failed or timed out, or if a synchronous resolution was canceled.

Discussion
This function updates the specified CFNetService with the IP address or addresses associated with the service. Call `CFNetServiceGetAddressing` (page 88) to get the addresses.

When resolving a service that runs in asynchronous mode, this function returns `TRUE` if the CFNetService has a domain, type, and name, and the underlying resolution process was started. Otherwise, this function returns `FALSE`. Once started, the resolution continues until it is canceled by calling `CFNetServiceCancel` (page 84).

When resolving a service that runs in synchronous mode, this function blocks until the CFNetService is updated with at least one IP address, until an error occurs, or until `CFNetServiceCancel` (page 84) is called.

Special Considerations

This function is thread safe.

If the service will be used in asynchronous mode, you must call `CFNetServiceSetClient` (page 102) before calling this function.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFNetServices.h`

CFNetServiceScheduleWithRunLoop

Schedules a CFNetService on a run loop.

```
void CFNetServiceScheduleWithRunLoop (
   CFNetServiceRef theService,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

theService
The CFNetService that is to be scheduled on a run loop; cannot be `NULL`.

runLoop
The run loop on which the service is to be scheduled; cannot be `NULL`.

runLoopMode
The mode on which to schedule the service; cannot be `NULL`.

Discussion
Schedules the specified service on a run loop, which places the service in asynchronous mode. The caller is responsible for ensuring that at least one of the run loops on which the service is scheduled is being run.

Special Considerations

This function is thread safe.

Before calling this function, call `CFNetServiceSetClient` (page 102) to prepare a CFNetService for use in asynchronous mode.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetServices.h`

CFNetServiceSetClient

Associates a callback function with a CFNetService or disassociates a callback function from a CFNetService.

```
Boolean CFNetServiceSetClient (
   CFNetServiceRef theService,
   CFNetServiceClientCallBack clientCB,
   CFNetServiceClientContext *clientContext
);
```

Parameters

theService
The CFNetService; cannot be `NULL`.

clientCB
The callback function that is to be associated with this CFNetService. If you are shutting down the service, set `clientCB` to `NULL` to disassociate from this CFNetService the callback function that was previously associated.

clientContext
Context information to be used when `clientCB` is called; cannot be `NULL`.

Return Value
`TRUE` if the client was set; otherwise, `FALSE`.

Discussion
The callback function specified by `clientCB` will be called to report IP addresses (in the case of `CFNetServiceResolve`) or to report registration errors (in the case of `CFNetServiceRegister`).

Special Considerations

This function is thread safe.

For a CFNetService that will operate asynchronously, call this function and then call `CFNetServiceScheduleWithRunLoop` (page 101) to schedule the service on a run loop. Then call `CFNetServiceRegister` (page 97) or `CFNetServiceResolve` (page 99).

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetServices.h`

CFNetServiceSetProtocolSpecificInformation

Sets protocol-specific information for a CFNetService. (Deprecated in Mac OS X version 10.4. Use `CFNetServiceSetTXTData` instead.)

```
void CFNetServiceSetProtocolSpecificInformation (
   CFNetServiceRef theService,
   CFStringRef theInfo
);
```

Parameters

theService

The CFNetService whose protocol-specific information is to be set; cannot be `NULL`.

theInfo

The protocol-specific information to be set. Pass `NULL` to remove protocol-specific information from the service.

Discussion

The protocol-specific information appears in DNS TXT records for the service. Each TXT record consists of zero or more strings, packed together without any intervening gaps or padding bytes for word alignment. The format of each constituent string is a single length byte, followed by zero to 255 bytes of text data.

Special Considerations

This function is thread safe.

Availability

Available in Mac OS X version 10.2 and later.

Deprecated in Mac OS X version 10.4.

Declared In

`CFNetServices.h`

CFNetServiceSetTXTData

Sets the TXT record for a CFNetService.

```
Boolean CFNetServiceSetTXTData (
   CFNetServiceRef theService,
   CFDataRef txtRecord
);
```

Parameters

theService

CFNetServiceRef for which a TXT record is to be set; cannot be `NULL`.

txtRecord

Contents of the TXT record that is to be set. The contents must not exceed 1450 bytes.

Return Value

`TRUE` if the TXT record was set; otherwise, `FALSE`.

Discussion

This function sets a TXT record for the specified service. If the service is currently registered on the network, the record is broadcast. Setting a TXT record on a service that is still being resolved is not allowed.

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFNetServices.h`

CFNetServiceUnscheduleFromRunLoop

Unschedules a CFNetService from a run loop.

```
void CFNetServiceUnscheduleFromRunLoop (
   CFNetServiceRef theService,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

`theService`
The CFNetService that is to be unscheduled; cannot be `NULL`.

`runLoop`
The run loop; cannot be `NULL`.

`runLoopMode`
The mode from which the service is to be unscheduled; cannot be `NULL`.

Discussion
Unschedules the specified service from the specified run loop and mode. Call this function to shut down a service that is running asynchronously. To complete the shutdown, call `CFNetServiceSetClient` (page 102) and set `clientCB` to `NULL`. Then call `CFNetServiceCancel` (page 84).

Special Considerations

This function is thread safe.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetServices.h`

Callbacks

CFNetServiceBrowserClientCallBack

Defines a pointer to the callback function for a CFNetServiceBrowser.

```
typedef void (*CFNetServiceBrowserClientCallBack) (
   CFNetServiceBrowserRef browser,
   CFOptionFlags flags,
   CFTypeRef domainOrService,
   CFStreamError* error,
   void* info);
```

If you name your callback `MyNetServiceBrowserClientCallBack`, you would declare it like this:

```
void MyNetServiceBrowserClientCallBack (
   CFNetServiceBrowserRef browser,
   CFOptionFlags flags,
   CFTypeRef domainOrService,
   CFStreamError* error,
   void* info);
```

Parameters

browser

The CFNetServiceBrowser associated with this callback function.

flags

Flags conveying additional information. The `kCFNetServiceFlagIsDomain` bit is set if `domainOrService` contains a domain; if this bit is not set, `domainOrService` contains a CFNetService instance. For additional bit values, see CFNetServiceBrowserClientCallBack Bit Flags (page 109).

domainOrService

A string containing a domain name if this callback function is being called as a result of calling `CFNetServiceBrowserSearchForDomains` (page 80), or a CFNetService instance if this callback function is being called as a result calling `CFNetServiceBrowserSearchForServices` (page 81).

error

A pointer to a `CFStreamError` structure whose `error` field may contain an error code.

info

User-defined context information. The value of `info` is the same as the value of the `info` field of the `CFNetServiceClientContext` (page 107) structure that was provided when `CFNetServiceBrowserCreate` (page 78) was called to create the CFNetServiceBrowser associated with this callback function.

Discussion

The callback function for a CFNetServiceBrowser is called one or more times when domains or services are found as the result of calling `CFNetServiceBrowserSearchForDomains` (page 80) and `CFNetServiceBrowserSearchForServices` (page 81).

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CFNetServices.h`

CFNetServiceClientCallBack

Defines a pointer to the callback function for a CFNetService.

```
typedef void (*CFNetServiceClientCallBack) (
   CFNetServiceRef theService,
   CFStreamError* error,
   void* info);
```

If you name your callback `MyNetServiceClientCallBack`, you would declare it like this:

```
void MyNetServiceClientCallBack (
   CFNetServiceRef theService,
   CFStreamError* error,
   void* info);
```

Parameters

theService

CFNetService associated with this callback function.

error

Pointer to a `CFStreamError` structure whose `error` field contain may contain an error code.

info

User-defined context information. The value of `info` is the same as the value of the `info` field of the `CFNetServiceClientContext` (page 107) structure that was provided when `CFNetServiceSetClient` (page 102) was called for the CFNetService associated with this callback function.

Discussion

Your callback function will be called when there are results of resolving a CFNetService to report or when there are registration errors to report. In the case of resolution, if the service has more than one IP address, your callback will be called once for each address.

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CFNetServices.h`

CFNetServiceMonitorClientCallBack

Defines a pointer to the callback function that is to be called when a monitored record type changes.

```
typedef void (*CFNetServiceMonitorClientCallBack) (
   CFNetServiceMonitorRef theMonitor,
   CFNetServiceRef theService,
   CFNetServiceMonitorType typeInfo,
   CFDataRef rdata,
   CFStreamError* error,
   void* info);
```

If you name your callback `MyNetServiceMonitorClientCallBack`, you would declare it like this:

```
void MyNetServiceMonitorClientCallBack (
   CFNetServiceMonitorRef theMonitor,
   CFNetServiceRef theService,
   CFNetserviceMonitorType typeInfo,
   CFDataRef rdata,
   CFStreamError *error,
```

```
    void *info);
```

Parameters

theMonitor
CFNetServiceMonitor for which the callback is being called.

theService
CFNetService for which the callback is being called.

typeInfo
Type of record that changed. For possible values, see CFNetServiceMonitorType Constants (page 110).

rdata
Contents of the record that changed.

error
Pointer to `CFStreamError` structure whose `error` field contains an error code if an error occurred.

info
Arbitrary pointer to the user-defined data that was specified in the `info` field of the `CFNetServiceClientContext` structure when the monitor was created by `CFNetServiceMonitorCreate` (page 92).

Discussion
The callback function will be called when the monitored record type changes or when the monitor is stopped by calling `CFNetServiceMonitorStop` (page 96).

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetServices.h`

Data Types

CFNetServiceBrowserRef

An opaque reference representing a CFNetServiceBrowser.

```
typedef struct __CFNetServiceBrowser* CFNetServiceBrowserRef;
```

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetServices.h`

CFNetServiceClientContext

A structure provided when a CFNetService is associated with a callback function or when a CFNetServiceBrowser is created.

```
struct CFNetServiceClientContext {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
};
typedef struct CFNetServiceClientContext CFNetServiceClientContext;
```

Fields

`version`

Version number for this structure. Currently the only valid value is zero.

`info`

Arbitrary pointer to user-allocated memory containing user-defined data that is associated with the service, browser, or monitor and is passed to their respective callback functions. The data must be valid for as long as the CFNetService, CFNetServiceBrowser, or CFNetServiceMonitor is valid. Set this field to `NULL` if your callback function doesn't want to receive user-defined data.

`retain`

The callback used to add a retain for the service or browser using `info` for the life of the service or browser. This callback may be used for temporary references the service or browser needs to take. This callback returns the actual `info` pointer so it can be stored in the service or browser. This field can be `NULL`.

`release`

Callback that removes a retain previously added for the service or browser on the `info` pointer. This field can be `NULL`, but setting this field to `NULL` may result in memory leaks.

`copyDescription`

Callback used to create a descriptive string representation of the data pointed to by `info`. In implementing this function, return a reference to a CFString object that describes your allocator and some characteristics of your user-defined data, which is used by `CFCopyDescription()`. You can set this field to `NULL`, in which case Core Foundation will provide a rudimentary description.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetServices.h`

CFNetServiceMonitorRef

An opaque reference for a service monitor.

```
typedef struct __CFNetServiceMonitor* CFNetServiceMonitorRef;
```

Discussion
Service monitor references are used to monitor record changes on a CFNetServiceRef.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`CFNetServices.h`

CFNetServiceRef

An opaque reference representing a CFNetService.

```
typedef struct __CFNetService* CFNetServiceRef;
```

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetServices.h`

Constants

CFNetService Registration Options

Bit flags used when registering a service.

```
enum {
    kCFNetServiceFlagNoAutoRename = 1
};
```

Constants
`kCFNetServiceFlagNoAutoRename`
Causes registrations to fail if a name conflict occurs.
Available in Mac OS X v10.4 and later.
Declared in `CFNetServices.h`.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetwork/CFNetServices.h`

CFNetServiceBrowserClientCallBack Bit Flags

Bit flags providing additional information about the result returned when a client's `CFNetServiceBrowserClientCallBack` function is called.

```
enum {
kCFNetServiceFlagMoreComing = 1,
kCFNetServiceFlagIsDomain = 2,
kCFNetServiceFlagIsDefault = 4,
kCFNetServiceFlagIsRegistrationDomain = 4, /* For compatibility  */
kCFNetServiceFlagRemove = 8
};
```

Constants

`kCFNetServiceFlagMoreComing`

If set, a hint that the client's callback function will be called again soon; therefore, the client should not do anything time-consuming, such as updating the screen.

Available in Mac OS X v10.2 and later.

Declared in `CFNetServices.h`.

`kCFNetServiceFlagIsDomain`

If set, the results pertain to a search for domains. If not set, the results pertain to a search for services.

Available in Mac OS X v10.2 and later.

Declared in `CFNetServices.h`.

`kCFNetServiceFlagIsDefault`

If set, the resulting domain is the default registration or browse domain, depending on the context. For this version of the CFNetServices API, the default registration domain is the local domain. In previous versions of this API, this constant was `kCFNetServiceFlagIsRegistrationDomain`, which is retained for backward compatibility.

Available in Mac OS X v10.4 and later.

Declared in `CFNetServices.h`.

`kCFNetServiceFlagRemove`

If set, the client should remove the result item instead of adding it.

Available in Mac OS X v10.2 and later.

Declared in `CFNetServices.h`.

Discussion

See `CFNetServiceBrowserClientCallBack` for additional information.

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CFNetwork/CFNetServices.h`

CFNetServiceMonitorType Constants

Record type specifier used to tell a service monitor the type of record changes to watch for.

```
enum {
kCFNetServiceMonitorTXT = 1
} typedef enum CFNetServiceMonitorType CFNetServiceMonitorType;
```

Constants

`kCFNetServiceMonitorTXT`

Watch for TXT record changes.

Available in Mac OS X v10.4 and later.

Declared in `CFNetServices.h`.

Availability

Available in Mac OS X version 10.2 and later.

Declared In

`CFNetwork/CFNetServices.h`

CFNetService Error Constants

Error codes that may be returned by CFNetServices functions or passed to CFNetServices callback functions.

```
typedef enum {
    kCFNetServicesErrorUnknown = -72000,
    kCFNetServicesErrorCollision = -72001,
    kCFNetServicesErrorNotFound = -72002,
    kCFNetServicesErrorInProgress = -72003,
    kCFNetServicesErrorBadArgument = -72004,
    kCFNetServicesErrorCancel = -72005,
    kCFNetServicesErrorInvalid = -72006,
    kCFNetServicesErrorTimeout = -72007
} CFNetServicesError;
```

Constants

`kCFNetServicesErrorUnknown`

An unknown CFNetService error occurred.

Available in Mac OS X v10.2 and later.

Declared in `CFNetServices.h`.

`kCFNetServicesErrorCollision`

An attempt was made to use a name that is already in use.

Available in Mac OS X v10.2 and later.

Declared in `CFNetServices.h`.

`kCFNetServicesErrorNotFound`

Not used.

Available in Mac OS X v10.2 and later.

Declared in `CFNetServices.h`.

`kCFNetServicesErrorInProgress`

A search is already in progress.

Available in Mac OS X v10.2 and later.

Declared in `CFNetServices.h`.

`kCFNetServicesErrorBadArgument`
A required argument was not provided.

Available in Mac OS X v10.2 and later.

Declared in `CFNetServices.h`.

`kCFNetServicesErrorCancel`
The search or service was canceled.

Available in Mac OS X v10.2 and later.

Declared in `CFNetServices.h`.

`kCFNetServicesErrorInvalid`
Invalid data was passed to a CFNetServices function.

Available in Mac OS X v10.2 and later.

Declared in `CFNetServices.h`.

`kCFNetServicesErrorTimeout`
Resolution failed because the timeout was reached.

Available in Mac OS X v10.4 and later.

Declared in `CFNetServices.h`.

Availability
Available in Mac OS X version 10.2 and later.

Declared In
`CFNetwork/CFNetServices.h`

Error Domains

Error domains.

```
extern const SInt32 kCFStreamErrorDomainMach;
extern const SInt32 kCFStreamErrorDomainNetServices;
```

Constants
`kCFStreamErrorDomainMach`
Error domain returning errors reported by Mach. For more information, see the header file `/usr/include/mach/error.h`.

Available in Mac OS X version 10.5 and later.

Declared in `CFNetServices.h`.

`kCFStreamErrorDomainNetServices`
Error domain returning errors reported by the service discovery APIs. These errors are only returned if you use the `CFNetServiceBrowser` API or any APIs introduced in Mac OS X v10.4 or later.

Available in Mac OS X version 10.5 and later.

Declared in `CFNetServices.h`.

CFStream Socket Additions

Derived From:	CFType
Framework:	CoreServices
Declared in	CFNetwork/CFSocketStream.h
Companion guide	CFNetwork Programming Guide

Overview

This document describes the `CFStream` functions for working with sockets. It is part of the CFSocketStream API.

Functions by Task

Creating Socket Pairs

`CFStreamCreatePairWithSocketToCFHost` (page 116)
Creates readable and writable streams connected to a given `CFHost` object.

`CFStreamCreatePairWithSocketToNetService` (page 116)
Creates a pair of streams for a CFNetService.

Setting the Security Protocol

`CFSocketStreamPairSetSecurityProtocol` (page 114)
This function sets the security protocol for the specified pair of socket streams. (Deprecated. Use `CFReadStreamSetProperty` and `CFWriteStreamSetProperty` in conjunction with the security constants defined in `CFSocketStream`.)

Obtaining Errors

`CFSocketStreamSOCKSGetError` (page 114)
This function gets error codes in the `kCFStreamErrorDomainSOCKS` domain from the `CFStreamError` returned by a stream operation.

`CFSocketStreamSOCKSGetErrorSubdomain` (page 115)
Gets the error subdomain associated with errors in the `kCFStreamErrorDomainSOCKS` domain from the `CFStreamError` returned by a stream operation.

Functions

CFSocketStreamPairSetSecurityProtocol

This function sets the security protocol for the specified pair of socket streams. (Deprecated in Mac OS X v10.2. Use `CFReadStreamSetProperty` and `CFWriteStreamSetProperty` in conjunction with the security constants defined in `CFSocketStream`.)

```
Boolean CFSocketStreamPairSetSecurityProtocol (
    CFReadStreamRef socketReadStream,
    CFWriteStreamRef socketWriteStream,
    CFStreamSocketSecurityProtocol securityProtocol
);
```

Parameters

socketReadStream
The read stream.

socketWriteStream
The write stream.

securityProtocol
The security protocol to be set. See CFStream Socket Security Protocol Constants (page 121) for possible values.

function result
`TRUE` if specified security protocol was set; otherwise, `FALSE`.

Discussion

Call this function before you call `CFReadStreamOpen` to open the read stream or `CFWriteStreamOpen` to open the write stream.

Special Considerations

This function is thread safe.

Availability

Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.2.

Declared In

`CFSocketStream.h`

CFSocketStreamSOCKSGetError

This function gets error codes in the `kCFStreamErrorDomainSOCKS` domain from the `CFStreamError` returned by a stream operation.

```
SInt32 CFSocketStreamSOCKSGetError(CFStreamError* error);
```

Parameters

error

The error value to decode.

Discussion

Error codes in the `kCFStreamErrorDomainSOCKS` domain can come from multiple parts of the protocol stack, many of which define their own error values as part of outside specifications such as the HTTP specification.

To avoid confusion from conflicting error numbers, error codes in the `kCFStreamErrorDomainSOCKS` domain contain two parts: a subdomain, which tells which part of the protocol stack generated the error, and the error code itself.

Calling `CFSocketStreamSOCKSGetError` (page 114) returns the error code itself, which must be interpreted in the context of the result of a call to `CFSocketStreamSOCKSGetErrorSubdomain` (page 115). Possible return values (beyond subdomain-specific values such as client versions and HTTP error codes) are listed in "CFStream Errors" (page 126).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`CFSocketStream.h`

CFSocketStreamSOCKSGetErrorSubdomain

Gets the error subdomain associated with errors in the `kCFStreamErrorDomainSOCKS` domain from the `CFStreamError` returned by a stream operation.

```
SInt32 CFSocketStreamSOCKSGetErrorSubdomain(CFStreamError* error);
```

Parameters

error

The error value to decode.

Discussion

Error codes in the `kCFStreamErrorDomainSOCKS` domain can come from multiple parts of the protocol stack, many of which define their own error values as part of outside specifications such as the HTTP specification.

To avoid confusion from conflicting error numbers, error codes in the `kCFStreamErrorDomainSOCKS` domain contain two parts: a subdomain, which tells which part of the protocol stack generated the error, and the error code itself.

Calling `CFSocketStreamSOCKSGetErrorSubdomain` (page 115) returns an identifier that tells which layer of the protocol stack produced the error. The possible values are listed in "Error Subdomains" (page 125). With this information, you can interpret the error codes returned by `CFSocketStreamSOCKSGetError` (page 114).

Availability

Available in Mac OS X v10.2 and later.

Declared In
CFSocketStream.h

CFStreamCreatePairWithSocketToCFHost

Creates readable and writable streams connected to a given CFHost object.

```
void CFStreamCreatePairWithSocketToCFHost (
   CFAllocatorRef alloc,
   CFHostRef host,
   SInt32 port,
   CFReadStreamRef *readStream,
   CFWriteStreamRef *writeStream
);
```

Parameters

alloc

The allocator to use to allocate memory for the CFReadStream and CFWriteStream objects. Pass NULL or kCFAllocatorDefault to use the current default allocator.

host

A CFHost object to which the streams are connected. If unresolved, the host will be resolved prior to connecting.

port

The TCP port number to which the socket streams should connect.

readStream

Upon return, contains a CFReadStream object connected to the host *host* on port port, or NULL if there is a failure during creation. If you pass NULL, the function will not create a readable stream. Ownership follows the Create Rule.

writeStream

Upon return, contains a CFWriteStream object connected to the host *host* on port port, or NULL if there is a failure during creation. If you pass NULL, the function will not create a writable stream. Ownership follows the Create Rule.

Discussion
The streams do not open a connection to the specified host until one of the streams is opened.

Most properties are shared by both streams. Setting the property for one stream automatically sets the property for the other.

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

Declared In
CFSocketStream.h

CFStreamCreatePairWithSocketToNetService

Creates a pair of streams for a CFNetService.

```
void CFStreamCreatePairWithSocketToNetService (
   CFAllocatorRef alloc,
   CFNetServiceRef service,
   CFReadStreamRef *readStream,
   CFWriteStreamRef *writeStream
);
```

Parameters

alloc

The allocator to use to allocate memory for the `CFReadStream` and `CFWriteStream` objects. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

service

Reference to the `CFNetService` to which the streams are to be connected. If the service is not resolved, the service will be resolved before the streams are connected.

readstream

Upon return, contains a `CFReadStream` object connected to the service specified by *service*, or `NULL` if there is a failure during creation. If you pass `NULL`, the function will not create a readable stream. Ownership follows the Create Rule.

writeStream

Upon return, contains a `CFWriteStream` object connected to the service specified by *service*, or `NULL` if there is a failure during creation. If you pass `NULL`, the function will not create a writable stream. Ownership follows the Create Rule.

Discussion

Read and write operations on sockets can block. To prevent blocking, you can call `CFReadStreamSetClient` and `CFWriteStreamSetClient` to register to receive stream-related event notifications. Then call `CFReadStreamScheduleWithRunLoop` and `CFWriteStreamScheduleWithRunLoop` to schedule the stream on a run loop for receiving stream-related event notifications. Then call `CFReadStreamOpen` and `CFWriteStreamOpen` to open each stream.

Special Considerations

This function is thread safe.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CFSocketStream.h`

Constants

CFStream Property Keys

Constants for `CFStream` property keys

```
const CFStringRef kCFStreamPropertyShouldCloseNativeSocket;
const CFStringRef kCFStreamPropertySocketSecurityLevel;
const CFStringRef kCFStreamPropertySOCKSProxy;
const CFStringRef kCFStreamPropertySSLPeerCertificates;
const CFStringRef kCFStreamPropertySSLSettings;
const CFStringRef kCFStreamPropertyProxyLocalByPass;
extern const CFStringRef kCFStreamPropertySocketRemoteHost;
extern const CFStringRef kCFStreamPropertySocketRemoteNetService;
```

Constants

`kCFStreamPropertyShouldCloseNativeSocket`

Should Close Native Socket property key.

If set to `kCFBooleanTrue`, the stream will close and release the underlying native socket when the stream is released. If set to `kCFBooleanFalse`, the stream will not close and release the underlying native socket when the stream is released. If a stream is created with a native socket, the default value of this property is `kCFBooleanFalse`. This property is only available for socket streams. It can be set by calling `CFReadStreamSetProperty` and `CFWriteStreamSetProperty`, and it can be copied by `CFReadStreamCopyProperty` and `CFWriteStreamCopyProperty`.

Available in Mac OS X v10.2 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertySocketNativeHandle`

Socket Native Handle property key.

Causes `CFReadStreamCopyProperty` or `CFWriteStreamCopyProperty` to return `CFData` object that contains the native handle for a socket stream. This property is only available for socket streams.

Available in Mac OS X v10.1 and later.

Declared in `CFStream.h`.

`kCFStreamPropertySocketSecurityLevel`

Socket Security Level property key.

See CFStream Socket Security Level Constants (page 122) for specific security level constants to use.

Available in Mac OS X v10.2 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertySSLPeerCertificates`

SSL Peer Certificates property key for copy operations, which return a `CFArray` object containing `SecCertificateRef` objects.

For more information, see `SSLGetPeerCertificates` in `Security/SecureTransport.h`.

Available in Mac OS X v10.4 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertySOCKSProxy`

SOCKS proxy property key.

To set a `CFStream` object to use a SOCKS proxy, call `CFReadStreamSetProperty` or `CFWriteStreamSetProperty` with the property name set to `kCFStreamPropertySOCKSProxy` and its value set to a `CFDictionary` object having at minimum a `kCFStreamPropertySOCKSProxyHost` key and a `kCFStreamPropertySOCKSProxyPort` key. For information on these keys, see CFStream SOCKS Proxy Key Constants (page 123). SystemConfiguration returns a CFDictionary for SOCKS proxies that is usable without modification.

Available in Mac OS X v10.2 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertySSLSettings`
SSL Settings property key for set operations.

The key's value is a `CFDictionary` object containing security settings. For information on the dictionary's keys and values, see CFStream Property SSL Settings Constants (page 119). By default, there are no security settings.

Available in Mac OS X v10.4 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertyProxyLocalBypass`
Proxy Local Bypass property key.

The key's value is `CFBoolean` object whose value indicates whether local hostnames should be subject to proxy handling.

Available in Mac OS X v10.4 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertySocketRemoteHost`
The key's value is a `CFHostRef` for the remote host if it is known. If not, its value is `NULL`.

Available in Mac OS X version 10.3 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertySocketRemoteNetService`
The key's value is a `CFNetServiceRef` for the remote network service if it is known. If not, its value is `NULL`.

Available in Mac OS X version 10.3 and later.

Declared in `CFSocketStream.h`.

Declared In
`CFNetwork/CFSocketStream.h`

CFStream Property SSL Settings Constants

Constants for use in a `CFDictionary` object that is the value of the `kCFStreamPropertySSLSettings` stream property key.

```
const CFStringRef kCFStreamSSLLevel;
const CFStringRef kCFStreamSSLAllowsExpiredCertificates;
const CFStringRef kCFStreamSSLAllowsExpiredRoots;
const CFStringRef kCFStreamSSLAllowsAnyRoot;
const CFStringRef kCFStreamSSLValidatesCertificateChain;
const CFStringRef kCFStreamSSLPeerName;
const CFStringRef kCFStreamSSLCertificates;
const CFStringRef kCFStreamSSLIsServer;
```

Constants
`kCFStreamSSLLevel`
Security property key whose value specifies the stream's security level.

By default, a stream's security level is `kCFStreamSocketSecurityLevelNegotiatedSSL`. For other possible values, see CFStream Socket Security Level Constants (page 122).

Available in Mac OS X v10.4 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSSLAllowsExpiredCertificates`
: Security property key whose value indicates whether expired certificates are allowed.

  By default, the value of this key is `kCFBooleanFalse` (expired certificates are not allowed).

  Available in Mac OS X v10.4 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamSSLAllowsExpiredRoots`
: Security property key whose value indicates whether expired root certificates are allowed.

  By default, the value of this key is `kCFBooleanFalse` (expired root certificates are not allowed).

  Available in Mac OS X v10.4 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamSSLAllowsAnyRoot`
: Security property key whose value indicates whether root certificates should be allowed.

  By default, the value of this key is `kCFBooleanFalse` (root certificates are not allowed).

  Available in Mac OS X v10.4 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamSSLValidatesCertificateChain`
: Security property key whose value indicates whether the certificate chain should be validated.

  By default, the value of this key is `kCFBooleanTrue` (the certificate chain should be validated).

  Available in Mac OS X v10.4 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamSSLPeerName`
: Security property key whose value overrides the name used for certificate verification.

  By default, the host name that was used when the stream was created is used; if no host name was used, no peer name will be used. Set the value of this key to `kCFNull` to prevent name verification.

  Available in Mac OS X v10.4 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamSSLCertificates`
: Security property key whose value is a CFArray of SecCertificateRefs except for the first element in the array, which is a SecIdentityRef.

  For more information, see `SSLSetCertificate()` in `Security/SecureTransport.h`.

  Available in Mac OS X v10.4 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamSSLIsServer`
: Security property key whose value indicates whether the connection is to act as a server in the SSL process.

  By default, the value of this key is `kCFBooleanFalse` (the connection is not to act as a server). If the value of this key is `kCFBooleanTrue`, the `kCFStreamSSLCertificates` key must contain a valid value.

  Available in Mac OS X v10.4 and later.

  Declared in `CFSocketStream.h`.

Discussion
This enumeration defines the constants for keys in a `CFDictionary` object that is the value of the `kCFStreamPropertySSLSettings` key.

Declared In
`CFNetwork/CFSocketStream.h`

CFStream Socket Security Protocol Constants

Specifies constants for setting the security protocol for a socket stream.

```
typedef enum {
    kCFStreamSocketSecurityNone = 0,
    kCFStreamSocketSecuritySSLv2,
    kCFStreamSocketSecuritySSLv3,
    kCFStreamSocketSecuritySSLv23,
    kCFStreamSocketSecurityTLSv1
} CFStreamSocketSecurityProtocol;
```

Constants

`kCFStreamSocketSecurityNone`

Specifies that no security protocol be set for a socket stream. (**Deprecated.** Use `kCFStreamSocketSecurityLevelNone`.)

Available in Mac OS X v10.1 and later.

Deprecated in Mac OS X v10.2.

Declared in `CFSocketStream.h`.

`kCFStreamSocketSecuritySSLv2`

Specifies that SSL version 2 be set as the security protocol for a socket stream. (**Deprecated.** Use `kCFStreamSocketSecurityLevelSSLv2`.)

Available in Mac OS X v10.1 and later.

Deprecated in Mac OS X v10.2.

Declared in `CFSocketStream.h`.

`kCFStreamSocketSecuritySSLv3`

Specifies that SSL version 3 be set as the security protocol for a socket stream. (**Deprecated.** Use `kCFStreamSocketSecurityLevelSSLv3`.)

Available in Mac OS X v10.1 and later.

Deprecated in Mac OS X v10.2.

Declared in `CFSocketStream.h`.

`kCFStreamSocketSecuritySSLv23`

Specifies that SSL version 3 be set as the security protocol for a socket stream pair. If that version is not available, specifies that SSL version 2 be set as the security protocol for a socket stream. (**Deprecated.** Use `kCFStreamSocketSecurityLevelNegotiatedSSL`.)

Available in Mac OS X v10.1 and later.

Deprecated in Mac OS X v10.2.

Declared in `CFSocketStream.h`.

`kCFStreamSocketSecurityTLSv1`

Specifies that TLS version 1 be set as the security protocol for a socket stream. (**Deprecated.** Use `kCFStreamSocketSecurityLevelTLSv1`.)

Available in Mac OS X v10.1 and later.

Deprecated in Mac OS X v10.2.

Declared in `CFSocketStream.h`.

Discussion
This enumeration defines constants for setting the security protocol for a socket stream pair when calling `CFSocketStreamPairSetSecurityProtocol` (page 114).

Special Considerations

This enumeration is deprecated in favor of the constants described in CFStream Socket Security Level Constants (page 122).

Declared In
`CFNetwork/CFSocketStream.h`

CFStream Socket Security Level Constants

Constants for setting the security level of a socket stream.

```
const CFStringRef kCFStreamSocketSecurityLevelNone;
const CFStringRef kCFStreamSocketSecurityLevelSSLv2;
const CFStringRef kCFStreamSocketSecurityLevelSSLv3;
const CFStringRef kCFStreamSocketSecurityLevelTLSv1;
const CFStringRef kCFStreamSocketSecurityLevelNegotiatedSSL;
```

Constants

`kCFStreamSocketSecurityLevelNone`

Specifies that no security level be set.

Available in Mac OS X v10.2 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSocketSecurityLevelSSLv2`

Specifies that SSL version 2 be set as the security protocol for a socket stream.

Available in Mac OS X v10.2 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSocketSecurityLevelSSLv3`

Specifies that SSL version 3 be set as the security protocol for a socket stream pair.

If SSL version 3 is not available, specifies that SSL version 2 be set as the security protocol for a socket stream.

Available in Mac OS X v10.2 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSocketSecurityLevelTLSv1`

Specifies that TLS version 1 be set as the security protocol for a socket stream.

Available in Mac OS X v10.2 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSocketSecurityLevelNegotiatedSSL`
Specifies that the highest level security protocol that can be negotiated be set as the security protocol for a socket stream.

Available in Mac OS X v10.2 and later.

Declared in `CFSocketStream.h`.

Discussion
This enumeration defines the preferred constants for setting the security protocol for a socket stream pair when calling `CFReadStreamSetProperty` or `CFWriteStreamSetProperty`.

Declared In
`CFNetwork/CFSocketStream.h`

CFStream SOCKS Proxy Key Constants

Constants for SOCKS Proxy `CFDictionary` keys.

```
const CFStringRef kCFStreamPropertySOCKSProxyHost;
const CFStringRef kCFStreamPropertySOCKSProxyPort;
const CFStringRef kCFStreamPropertySOCKSVersion;
const CFStringRef kCFStreamSocketSOCKSVersion4;
const CFStringRef kCFStreamSocketSOCKSVersion5;
const CFStringRef kCFStreamPropertySOCKSUser;
const CFStringRef kCFStreamPropertySOCKSPassword;
```

Constants
`kCFStreamPropertySOCKSProxyHost`
Constant for the SOCKS proxy host key.

This key contains a CFString object that represents the SOCKS proxy host. Defined to match `kSCPropNetProxiesSOCKSProxy`.

Available in Mac OS X v10.2 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertySOCKSProxyPort`
Constant for the SOCKS proxy host port key.

This key contains a `CFNumberRef` object of type `kCFNumberSInt32Type` whose value represents the port on which the proxy listens.

Available in Mac OS X v10.2 and later.

Declared in `CFSocketStream.h`.

`kCFStreamPropertySOCKSVersion`
Constant for the SOCKS version key.

Its value must be `kCFStreamSocketSOCKSVersion4` or `kCFStreamSocketSOCKSVersion5` to set SOCKS4 or SOCKS5, respectively. If this key is not present, SOCKS5 is used by default.

Available in Mac OS X v10.2 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSocketSOCKSVersion4`
Constant used in the `kCFStreamSockerSOCKSVersion` key to specify SOCKS4 as the SOCKS version for the stream.

Available in Mac OS X v10.2 and later.

Declared in `CFSocketStream.h`.

`kCFStreamSocketSOCKSVersion5`
: Constant used in the `kCFStreamSOCKSVersion` key to specify SOCKS5 as the SOCKS version for the stream.

  Available in Mac OS X v10.2 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamPropertySOCKSUser`
: Constant for the key required to set a user name.

  The value is a `CFString` object containing the user's name.

  Available in Mac OS X v10.2 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamPropertySOCKSPassword`
: Constant for the key required to set a user's password.

  The value is a `CFString` object containing the user's password.

  Available in Mac OS X v10.2 and later.

  Declared in `CFSocketStream.h`.

Discussion
When setting the stream's SOCKS Proxy property, the property's value is a `CFDictionary` object containing at minimum the `kCFStreamPropertySOCKSProxyHost` and `kCFStreamPropertySOCKSProxyPort` keys. The dictionary may also contain the other keys described in this section.

Error Domains

Error domains specific to `CFSocketStream` calls.

```
extern const int kCFStreamErrorDomainSOCKS;
extern const int kCFStreamErrorDomainSSL;
extern const CFIndex kCFStreamErrorDomainWinSock;
```

Constants
`kCFStreamErrorDomainSOCKS`
: This domain returns error codes from the SOCKS layer. The errors are described in

  Available in Mac OS X version 10.5 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamErrorDomainSSL`
: This domain returns error codes associated with the SSL layer. For a list of error codes, see the header `SecureTransport.h` in `Security.framework`.

  Available in Mac OS X version 10.5 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamErrorDomainWinSock`
: When running CFNetwork code on Windows, this domain returns error codes associated with the underlying TCP/IP stack. You should also note that non-networking errors such as `ENOMEM` are delivered through the POSIX domain. See the header `winsock2.h` for relevant error codes.

  Available in Mac OS X version 10.5 and later.

  Declared in `CFSocketStream.h`.

Discussion
To determine the source of an error, examine the `userInfo` dictionary included in the `CFError` object returned by a function call or call `CFErrorGetDomain` and pass in the `CFError` object and the domain whose value you want to read.

Error Subdomains

Subdomains used to determine how to interpret an error in the `kCFStreamErrorDomainSOCKS` domain.

```
enum {
    kCFStreamErrorSOCKSSubDomainNone = 0,
    kCFStreamErrorSOCKSSubDomainVersionCode = 1,
    kCFStreamErrorSOCKS4SubDomainResponse = 2,
    kCFStreamErrorSOCKS5SubDomainUserPass = 3,
    kCFStreamErrorSOCKS5SubDomainMethod = 4,
    kCFStreamErrorSOCKS5SubDomainResponse = 5
};
```

Constants

`kCFStreamErrorSOCKSSubDomainNone`
The error code returned is a SOCKS error number.
Available in Mac OS X version 10.5 and later.

`kCFStreamErrorSOCKSSubDomainVersionCode`
The error returned contains the version of SOCKS that the server wishes to use.
Available in Mac OS X version 10.5 and later.

`kCFStreamErrorSOCKS4SubDomainResponse`
The error returned is the status code that the server returned after the last operation.
Available in Mac OS X version 10.5 and later.

`kCFStreamErrorSOCKS5SubDomainUserPass`
This subdomain returns error codes associated with the last authentication attempt.
Available in Mac OS X version 10.5 and later.

`kCFStreamErrorSOCKS5SubDomainMethod`
This subdomain returns the server's desired negotiation method.
Available in Mac OS X version 10.5 and later.

`kCFStreamErrorSOCKS5SubDomainResponse`
This subdomain returns the response code sent by the server when replying to a connection request.
Available in Mac OS X version 10.5 and later.

Discussion
Error codes in the `kCFStreamErrorDomainSOCKS` domain can come from multiple parts of the protocol stack, many of which define their own error values as part of outside specifications such as the HTTP specification.

To avoid confusion from conflicting error numbers, error codes in the `kCFStreamErrorDomainSOCKS` domain contain two parts: a subdomain, which tells which part of the protocol stack generated the error, and the error code itself.

Calling `CFSocketStreamSOCKSGetErrorSubdomain` (page 115) returns an identifier that tells which layer of the protocol stack produced the error. This list of constants contains the possible values that this function will return.

Calling `CFSocketStreamSOCKSGetError` (page 114) returns the actual error code that the subdomain describes.

CFStream Errors

Error codes returned by the `kCFStreamErrorDomainSOCKS` error domain.

```
/* kCFStreamErrorSOCKSSubDomainNone*/
enum {
  kCFStreamErrorSOCKS5BadResponseAddr = 1,
  kCFStreamErrorSOCKS5BadState  = 2,
  kCFStreamErrorSOCKSUnknownClientVersion = 3
};


/* kCFStreamErrorSOCKS4SubDomainResponse*/
enum {
  kCFStreamErrorSOCKS4RequestFailed = 91,
  kCFStreamErrorSOCKS4IdentdFailed = 92,
  kCFStreamErrorSOCKS4IdConflict = 93
};

/* kCFStreamErrorSOCKS5SubDomainMethod*/
enum {
  kSOCKS5NoAcceptableMethod     = 0xFF
};
```

Constants

`kCFStreamErrorSOCKS5BadResponseAddr`

The address returned is not of a known type. This error code is only valid for errors in the `kCFStreamErrorSOCKSSubDomainNone` subdomain.

Available in Mac OS X version 10.5 and later.

Declared in `CFSocketStream.h`.

`kCFStreamErrorSOCKS5BadState`

The stream is not in a state that allows the requested operation. This error code is only valid for errors in the `kCFStreamErrorSOCKSSubDomainNone` subdomain..

Available in Mac OS X version 10.5 and later.

Declared in `CFSocketStream.h`.

`kCFStreamErrorSOCKSUnknownClientVersion`

The SOCKS server rejected access because it does not support connections with the requested SOCKS version. SOCKS client version. You can query the `kCFSOCKSVersionKey` key to find out what version the server requested. This error code is only valid for errors in the `kCFStreamErrorSOCKSSubDomainNone` subdomain.

Available in Mac OS X version 10.5 and later.

Declared in `CFSocketStream.h`.

`kCFStreamErrorSOCKS4RequestFailed`
: Request rejected by the server or request failed. This error is specific to SOCKS4. This error code is only valid for errors in the `kCFStreamErrorSOCKS4SubDomainResponse` subdomain.

  Available in Mac OS X version 10.5 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamErrorSOCKS4IdentdFailed`
: Request rejected by the server because it could not connect to the `identd` daemon on the client. This error is specific to SOCKS4. This error code is only valid for errors in the `kCFStreamErrorSOCKS4SubDomainResponse` subdomain.

  Available in Mac OS X version 10.5 and later.

  Declared in `CFSocketStream.h`.

`kCFStreamErrorSOCKS4IdConflict`
: Request rejected by the server because the client program and the `identd` daemon reported different user IDs. This error is specific to SOCKS4. This error code is only valid for errors in the `kCFStreamErrorSOCKS4SubDomainResponse` subdomain.

  Available in Mac OS X version 10.5 and later.

  Declared in `CFSocketStream.h`.

`kSOCKS5NoAcceptableMethod`
: The client and server could not find a mutually agreeable authentication method. This error code is only valid for errors in the `kCFStreamErrorSOCKS5SubDomainMethod` subdomain.

  Available in Mac OS X version 10.5 and later.

  Declared in `CFSocketStream.h`.

Discussion

Error codes in the `kCFStreamErrorDomainSOCKS` domain can come from multiple parts of the protocol stack, many of which define their own error values as part of outside specifications such as the HTTP specification.

To avoid confusion from conflicting error numbers, error codes in the `kCFStreamErrorDomainSOCKS` domain contain two parts: a subdomain, which tells which part of the protocol stack generated the error, and the error code itself.

Calling `CFSocketStreamSOCKSGetErrorSubdomain` (page 115) returns an identifier that tells which layer of the protocol stack produced the error.

Calling `CFSocketStreamSOCKSGetError` (page 114) returns the actual error code that the subdomain describes. This list of constants contains the possible values that this function will return. They must be interpreted within the context of the relevant error subdomain.

MDItem Reference

Derived From:	CFType
Framework:	CoreServices/CoreServices.h
Declared in	MDItem.h
Companion guides	Spotlight Overview Spotlight Query Programming Guide Spotlight Importer Programming Guide Spotlight Metadata Attributes Reference

Overview

MDItem is a CF-compliant object that represents a file and the metadata associated with the file.

For functions that expect an MDItemRef parameter, if this parameter is not a valid MDItemRef, the behavior is undefined. `NULL` is not a valid MDItemRef.

Functions by Task

Creating an MDItem

`MDItemCreate` (page 132)
Creates an MDItem object for a file at the specified path.

Getting the Type Identifier

`MDItemGetTypeID` (page 132)
Returns the type identifier of all MDItem instances.

Retrieving Metadata Attributes

`MDItemCopyAttribute` (page 130)
Returns the value of the specified attribute in the metadata item.

`MDItemCopyAttributes` (page 131)
Returns the values of the specified attributes in the metadata item.

`MDItemCopyAttributeList` (page 130)
Returns the values of the specified attributes in the metadata item.

`MDItemCopyAttributeNames` (page 131)
Returns an array containing the attribute names existing in the metadata item.

Functions

MDItemCopyAttribute

Returns the value of the specified attribute in the metadata item.

```
CFTypeRef MDItemCopyAttribute (
   MDItemRef item,
   CFStringRef name
);
```

Parameters

item
The item to be queried.

name
The name of the requested attribute.

Return Value

A CFTypeRef, or `NULL` if there was a failure reading the attribute or the attribute does not exist.

Availability

Available in Mac OS X version 10.4 and later.

Declared In

`MDItem.h`

MDItemCopyAttributeList

Returns the values of the specified attributes in the metadata item.

```
CFDictionaryRef MDItemCopyAttributeList (
   MDItemRef item,
   ...
);
```

Parameters

item
The item to be queried.

...
A comma-separated varargs list of the string attribute names..

Return Value

A CFDictionary containing keys for the requested attribute names, and the corresponding values. If an attribute does not exist, or the attribute is unreadable, there will be no key-value pair for it in the dictionary. Returns `NULL` on failure.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`MDItem.h`

MDItemCopyAttributeNames

Returns an array containing the attribute names existing in the metadata item.

```
CFArrayRef MDItemCopyAttributeNames (
   MDItemRef item
);
```

Parameters

item

The item to be queried.

Return Value
A CFArray of CFString attribute names, or `NULL` on failure.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`MDItem.h`

MDItemCopyAttributes

Returns the values of the specified attributes in the metadata item.

```
CFDictionaryRef MDItemCopyAttributes (
   MDItemRef item,
   CFArrayRef names
);
```

Parameters

item

The item to be queried.

names

A CFArray containing the names of the requested attributes.

Return Value
A CFDictionary containing keys for the requested attribute names, and the corresponding values. If an attribute does not exist, or the attribute is unreadable, there will be no key-value pair for it in the dictionary. Returns `NULL` on failure.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`MDItem.h`

MDItemCreate

Creates an MDItem object for a file at the specified path.

```
MDItemRef MDItemCreate (
   CFAllocatorRef allocator,
   CFStringRef path
);
```

Parameters

allocator

The `CFAllocator` object to be used to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

path

A path to the file from which to create the `MDItem`. The path must exist.

Return Value
An `MDItem` object or `NULL` if there was a problem creating the object.

Discussion
Returns a metadata item for the given path. MDItemRefs are uniqued and can be compared using == or `CFEqual`.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`MDItem.h`

MDItemGetTypeID

Returns the type identifier of all MDItem instances.

```
CFTypeID MDItemGetTypeID (
   void
);
```

Return Value
The type identifier for the `MDItem` opaque type.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`MDItem.h`

Data Types

MDItemRef

A reference to a MDItem object.

```
typedef struct __MDItem *MDItemRef;
```

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

Declared In
`MDItem.h`

Constants

Common Metadata Attribute Keys

Metadata attribute keys that are common to many file types.

```
const CFStringRef kMDItemAttributeChangeDate;
const CFStringRef kMDItemAudiences;
const CFStringRef kMDItemAuthors;
const CFStringRef kMDItemCity;
const CFStringRef kMDItemComment;
const CFStringRef kMDItemContactKeywords;
const CFStringRef kMDItemContentCreationDate;
const CFStringRef kMDItemContentModificationDate;
const CFStringRef kMDItemContentType;
const CFStringRef kMDItemContributors;
const CFStringRef kMDItemCopyright;
const CFStringRef kMDItemCountry;
const CFStringRef kMDItemCoverage;
const CFStringRef kMDItemCreator;
const CFStringRef kMDItemDescription;
const CFStringRef kMDItemDueDate;
const CFStringRef kMDItemDurationSeconds;
const CFStringRef kMDItemEmailAddresses;
const CFStringRef kMDItemEncodingApplications;
const CFStringRef kMDItemFinderComment;
const CFStringRef kMDItemFonts;
const CFStringRef kMDItemHeadline;
const CFStringRef kMDItemIdentifier;
const CFStringRef kMDItemInstantMessageAddresses;
const CFStringRef kMDItemInstructions;
const CFStringRef kMDItemKeywords;
const CFStringRef kMDItemKind;
const CFStringRef kMDItemLanguages;
const CFStringRef kMDItemLastUsedDate;
const CFStringRef kMDItemNumberOfPages;
const CFStringRef kMDItemOrganizations;
const CFStringRef kMDItemPageHeight;
const CFStringRef kMDItemPageWidth;
const CFStringRef kMDItemPhoneNumbers;
const CFStringRef kMDItemProjects;
const CFStringRef kMDItemPublishers;
const CFStringRef kMDItemRecipients;
const CFStringRef kMDItemRights;
const CFStringRef kMDItemSecurityMethod;
const CFStringRef kMDItemStarRating;
const CFStringRef kMDItemStateOrProvince;
const CFStringRef kMDItemTextContent;
const CFStringRef kMDItemTitle;
const CFStringRef kMDItemVersion;
const CFStringRef kMDItemWhereFroms;
```

Constants

`kMDItemAttributeChangeDate`

The date and time of the last change made to a metadata attribute. A CFDate.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemAudiences`

The audience for which the file is intended. The audience may be determined by the creator or the publisher or by a third party. A CFArray of CFStrings.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemAuthors`
: The author, or authors, of the contents of the file. The order of the authors is preserved, but does not represent the main author or relative importance of the authors. A CFArray of CFStrings.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemCity`
: Identifies city of origin according to guidelines established by the provider. For example, "New York", "Cupertino", or "Toronto". A CFString.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemComment`
: A comment related to the file. This differs from the Finder comment, `kMDItemFinderComment`. A CFString.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemContactKeywords`
: A list of contacts that are associated with this document, not including the authors. A CFArray of CFStrings.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemContentCreationDate`
: The date that the contents of the file were created. This is different than the file creation date. Its can be used to store when the file contents were first created, or first modified. A CFDate.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemContentModificationDate`
: The date and time that the contents of the file were last modified. This is not necessarily the file modification date. A CFDate.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemContentType`
: The UTI pedigree of a file. For example, a jpeg image file will have a value of public.jpeg/public.image/public.data. The value of this attribute is set by the MDImporter. Changes to this value are lost when the file attributes are next imported. A CFString.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemContributors`
: The entities responsible for making contributions to the content of the resource. Examples of a contributor include a person, an organization or a service. A CFArray of CFStrings.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemCopyright`
The copyright owner of the file contents. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemCountry`
The full, publishable name of the country or primary location where the intellectual property of the item was created, according to guidelines of the provider. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemCoverage`
The extent or scope of the content of the resource. Coverage will typically include spatial location (a place name or geographic co-ordinates), temporal period (a period label, date, or date range) or jurisdiction (such as a named administrative entity). Recommended best practice is to select a value from a controlled vocabulary, and that, where appropriate, named places or time periods be used in preference to numeric identifiers such as sets of co-ordinates or date ranges. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemCreator`
Application used to create the document content (e.g. "Word", "AppleWorks", etc.). A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemDescription`
A description of the content of the resource. The description may include an abstract, table of contents, reference to a graphical representation of content or a free-text account of the content. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemDueDate`
Date this item is due. A CFDate.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemDurationSeconds`
The duration, in seconds, of the content of file. A value of 10.5 represents media that is 10 and 1/2 seconds long. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemEmailAddresses`
Email addresses related to this item. A CFArray of CFStrings.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemEncodingApplications`
Application used to convert the original content into it's current form. For example, a PDF file might have an encoding application set to "Distiller". A CFArray of CFStrings.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemFinderComment`
Finder comments for this file. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemFonts`
Fonts used in this item. You should store the font's full name, the postscript name, or the font family name, based on the available information. A CFArray of CFStrings.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemHeadline`
A publishable entry providing a synopsis of the contents of the file. For example, "Apple Introduces the iPod Photo". A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemIdentifier`
A formal identifier used to reference the resource within a given context. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemInstantMessageAddresses`
Instant message addresses related to this item. A CFArray of CFStrings.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemInstructions`
Editorial instructions concerning the use of the item, such as embargoes and warnings. For example, "Second of four stories". A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemKeywords`
Keywords associated with this file. For example, “Birthday”, “Important”, etc. A CFArray of CFStrings.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemKind`
A description of the kind of item this file represents. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemLanguages`
Indicates the languages of the intellectual content of the resource. Recommended best practice for the values of the Language element is defined by RFC 3066. A CFArray of CFStrings.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemLastUsedDate`
: The date and time that the file was last used. This value is updated automatically by LaunchServices everytime a file is opened by double clicking, or by asking LaunchServices to open a file. A CFDate.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemNumberOfPages`
: Number of pages in the document. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemOrganizations`
: The company or organization that created the document. A CFArray of CFStrings.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemPageHeight`
: Height of the document page, in points (72 points per inch). For PDF files this indicates the height of the first page only. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemPageWidth`
: Width of the document page, in points (72 points per inch). For PDF files this indicates the width of the first page only. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemPhoneNumbers`
: Phone numbers related to this item. A CFArray of CFStrings.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemProjects`
: The list of projects that this file is part of. For example, if you were working on a movie all of the files could be marked as belonging to the project “My Movie”. A CFArray of CFStrings.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemPublishers`
: The entity responsible for making the resource available. For example, a person, an organization, or a service. Typically, the name of a publisher should be used to indicate the entity. A CFArray of CFStrings.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemRecipients`
: Recipients of this item. A CFArray of CFStrings.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemRights`
: Provides a link to information about rights held in and over the resource. Contains a rights management statement for the resource, or reference a service providing such information. Rights information often encompasses Intellectual Property Rights (IPR), Copyright, and various Property Rights. If this attribute is absent, no assumptions can be made about the status of these and other rights with respect to the resource. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemSecurityMethod`
: The security or encryption method used for the file. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemStarRating`
: User rating of this item. For example, the stars rating of an iTunes track. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemStateOrProvince`
: Identifies the province or state of origin according to guidelines established by the provider. For example, "CA", "Ontario", or "Sussex". A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemTextContent`
: Contains a text representation of the content of the document. Data in multiple fields should be combined using a whitespace character as a separator. An application's Spotlight importer provides the content of this attribute. Applications can search for values in this attribute, but are not able to read the content of this attribute directly. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemTitle`
: The title of the file. For example, this could be the title of a document, the name of an song, or the subject of an email message. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemVersion`
: The version number of this file. A CFString

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemWhereFroms`
: Describes where the file was obtained from. For example, a downloaded file may refer to the URL, files received by email may indicate the sender's email address, message subject, etc. A CFArray of CFStrings.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`MDItem.h`

Image Metadata Attribute Keys

Metadata attribute keys that are common to image files.

```
const CFStringRef      kMDItemPixelHeight;
const CFStringRef      kMDItemPixelWidth;
const CFStringRef      kMDItemColorSpace;
const CFStringRef      kMDItemBitsPerSample;
const CFStringRef      kMDItemFlashOnOff;
const CFStringRef      kMDItemFocalLength;
const CFStringRef      kMDItemAcquisitionMake;
const CFStringRef      kMDItemAcquisitionModel;
const CFStringRef      kMDItemISOSpeed;
const CFStringRef      kMDItemOrientation;
const CFStringRef      kMDItemLayerNames;
const CFStringRef      kMDItemWhiteBalance;
const CFStringRef      kMDItemAperture;
const CFStringRef      kMDItemProfileName;
const CFStringRef      kMDItemResolutionWidthDPI;
const CFStringRef      kMDItemResolutionHeightDPI;
const CFStringRef      kMDItemExposureMode;
const CFStringRef      kMDItemExposureTimeSeconds;
const CFStringRef      kMDItemEXIFVersion;
const CFStringRef      kMDItemAlbum;
const CFStringRef      kMDItemHasAlphaChannel;
const CFStringRef      kMDItemRedEyeOnOff;
const CFStringRef      kMDItemMeteringMode;
const CFStringRef      kMDItemMaxAperture;
const CFStringRef      kMDItemFNumber;
const CFStringRef      kMDItemExposureProgram;
const CFStringRef      kMDItemExposureTimeString;
```

Constants

`kMDItemPixelHeight`
The height, in pixels, of the contents. For example, the image height or the video frame height. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemPixelWidth`
The width, in pixels, of the contents. For example, the image width or the video frame width. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemColorSpace`
The color space model used by the document contents. For example, “RGB”, “CMYK”, “YUV”, or “YCbCr”. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemBitsPerSample`
The number of bits per sample. For example, the bit depth of an image (8-bit, 16-bit etc...) or the bit depth per audio sample of uncompressed audio data (8, 16, 24, 32, 64, etc..). A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemFlashOnOff`
Indicates if a camera flash was used. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemFocalLength`
The actual focal length of the lens, in millimeters. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemAcquisitionMake`
The manufacturer of the device used to aquire the document contents. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemAcquisitionModel`
The model of the device used to aquire the document contents. For example, 100, 200, 400, etc. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemISOSpeed`
The ISO speed used to aquire the document contents. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemOrientation`
The orientation of the document contents. Possible values are 0 (landscape) and 1 (portrait). A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemLayerNames`
The names of the layers in the file. A CFArray of CFStrings.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemWhiteBalance`
The white balance setting used to aquire the document contents. Possible values are 0 (auto white balance) and 1 (manual). A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemAperture`
The aperature setting used to aqure the document contents. This unit is the APEX value. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemProfileName`
The name of the color profile used by the document contents. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemResolutionWidthDPI`
Resolution width, in DPI, of this image. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemResolutionHeightDPI`
Resolution height, in DPI, of this image. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemExposureMode`
The exposure mode used to aquire the document contents. Possible values are 0 (auto exposure), 1 (manual exposure) and 2 (auto bracket). A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemExposureTimeSeconds`
The exposure time, in seconds, used to aquire the document contents. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemEXIFVersion`
The version of the EXIF header used to generate the metadata. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemAlbum`
The title for a collection of media. This is analagous to a record album, or photo album. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemHasAlphaChannel`
Indicates if this image file has an alpha channel. A CFBoolean.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemRedEyeOnOff`
Indicates if red-eye reduction was used to take the picture. Possible values are 0 (no red-eye reduction mode or unknown) and 1 (red-eye reduction used). A CFBoolean.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemMeteringMode`
The metering mode used to take the image. Possible values are: Unknown, Average, CenterWeightedAverage, Spot, MultiSpot, Pattern, and Partial. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemMaxAperture`
The smallest f-number of the lens. The unit is the APEX?? value. Ordinarily it is given in the range of 00.00 to 99.99. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemFNumber`
The diameter of the diaphragm aperture in terms of the effective focal length of the lens.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemExposureProgram`
The class of the exposure program used by the camera to set exposure when the image is taken. Possible values include: Manual, Normal, and Aperture priority.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemExposureTimeString`
The time of the exposure. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`MDItem.h`

Video Metadata Attribute Keys

Metadata attribute keys that are common to video files.

```
const CFStringRef kMDItemAudioBitRate;
const CFStringRef kMDItemCodecs;
const CFStringRef kMDItemDeliveryType;
const CFStringRef kMDItemMediaTypes;
const CFStringRef kMDItemStreamable;
const CFStringRef kMDItemTotalBitRate;
const CFStringRef kMDItemVideoBitRate;
```

Constants
`kMDItemAudioBitRate`
The audio bit rate. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemCodecs`
: The codecs used to encode/decode the media. A CFArray of CFStrings.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemDeliveryType`
: The delivery type. Values are “Fast start” or “RTSP”. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemMediaTypes`
: The media types present in the content. A CFArray of CFStrings.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemStreamable`
: Whether the content is prepared for streaming. A CFBoolean.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemTotalBitRate`
: The total bit rate, audio and video combined, of the media. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemVideoBitRate`
: The video bit rate. A CFNumber.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`MDItem.h`

Audio Metadata Attribute Keys

Metadata attribute keys that describe an audio file.

```
const CFStringRef kMDItemAppleLoopDescriptors;
const CFStringRef kMDItemAppleLoopsKeyFilterType;
const CFStringRef kMDItemAppleLoopsLoopMode;
const CFStringRef kMDItemAppleLoopsRootKey;
const CFStringRef kMDItemAudioChannelCount;
const CFStringRef kMDItemAudioEncodingApplication;
const CFStringRef kMDItemAudioSampleRate;
const CFStringRef kMDItemAudioTrackNumber;
const CFStringRef kMDItemComposer;
const CFStringRef kMDItemIsGeneralMIDISequence;
const CFStringRef kMDItemKeySignature;
const CFStringRef kMDItemLyricist;
const CFStringRef kMDItemMusicalGenre;
const CFStringRef kMDItemMusicalInstrumentCategory;
const CFStringRef kMDItemMusicalInstrumentName;
const CFStringRef kMDItemRecordingDate;
const CFStringRef kMDItemRecordingYear;
const CFStringRef kMDItemTempo;
const CFStringRef kMDItemTimeSignature;
```

Constants

`kMDItemAppleLoopDescriptors`

Specifies multiple pieces of descriptive information about a loop. Besides genre and instrument, files can contain descriptive information that help users in refining searches. A CFArray of CFStrings.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemAppleLoopsKeyFilterType`

Specifies key filtering information about a loop. Loops are matched against projects that often in a major or minor key. To assist users in identifying loops that will "fit" with their compositions, loops can be tagged with one of the following key filters: "AnyKey" "Minor" "Major" "NeitherKey" "BothKeys". "AnyKey" means that it fits with anything (whether in a major key, minor key or neither). "Minor" fits with compositions in a minor key. "NeitherKey" doesn't work well with compositions that are in major or minor key. "BothKeys" means it fits with major or minor key. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemAppleLoopsLoopMode`

Specifies how a file should be played. Tagged files can either be loops or non-loops (e.g., a cymbal crash). "Looping" indicates if the file should be treated as a loop. "Non-looping" indicates the file should not be treated as a loop. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemAppleLoopsRootKey`

Specifies the loop's original key. The key is the root note or tonic for the loop, and does not include the scale type. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemAudioChannelCount`
: Number of channels in the audio data contained in the file. This integer value only represents the number of discreet channels of audio data found in the file. It does not indicate any configuration of the data in regards to a user's speaker setup. A CFNumber.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemAudioEncodingApplication`
: The name of the application that encoded the data contained in the audio file. A CFString.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemAudioSampleRate`
: Sample rate of the audio data contained in the file. The sample rate is a float value representing hz (audio_frames/second). For example: 44100.0, 22254.54. A CFNumber.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemAudioTrackNumber`
: The track number of a song or composition when it is part of an album. A CFNumber (integer).

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemComposer`
: The composer of the music contained in the audio file. A CFString.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemIsGeneralMIDISequence`
: Indicates whether the MIDI sequence contained in the file is setup for use with a General MIDI device. A CFBoolean.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemKeySignature`
: The key of the music contained in the audio file. For example: C, Dm, F#m, Bb. A CFString.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemLyricist`
: The lyricist, or text writer, of the music contained in the audio file. A CFString.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemMusicalGenre`
: The musical genre of the song or composition contained in the audio file. For example: Jazz, Pop, Rock, Classical. A CFString.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemMusicalInstrumentCategory`
: Specifies the category of an instrument. Files should have an instrument associated with them ("Other Instrument" is provided as a catch-all). For some categories, such as "Keyboards", there are instrument names which provide a more detailed instrument definition, for example "Piano" or "Organ". A CFString.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemMusicalInstrumentName`
: Specifies the name of instrument relative to the instrument category. Files can have an instrument name associated with them if they have certain instrument categories. For example, the "Percussion" category has multiple instruments, including "Conga" and "Bongo". A CFString.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemRecordingDate`
: The recording date of the song or composition. This is in contrast to `kMDItemContentCreationDate` which, could indicate the creation date of an edited or 'mastered' version of the original art. A CFDate.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemRecordingYear`
: Indicates the year the item was recorded. For example, 1964, 2003, etc. A CFNumber.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemTempo`
: A float value that specifies the beats per minute of the music contained in the audio file. A CFNumber.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemTimeSignature`
: The time signature of the musical composition contained in the audio/MIDI file. For example: "4/4", "7/8". A CFString.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`MDItem.h`

File System Metadata Attribute Keys

Metadata attribute keys that describe the file system attributes for a file.

```
const CFStringRef kMDItemDisplayName;
const CFStringRef kMDItemFSContentChangeDate;
const CFStringRef kMDItemFSCreationDate;
const CFStringRef kMDItemFSExists;
const CFStringRef kMDItemFSInvisible;
const CFStringRef kMDItemFSIsExtensionHidden;
const CFStringRef kMDItemFSIsReadable;
const CFStringRef kMDItemFSIsWriteable;
const CFStringRef kMDItemFSLabel;
const CFStringRef kMDItemFSName;
const CFStringRef kMDItemFSNodeCount;
const CFStringRef kMDItemFSOwnerGroupID;
const CFStringRef kMDItemFSOwnerUserID;
const CFStringRef kMDItemFSSize;
const CFStringRef kMDItemPath;
```

Constants

`kMDItemDisplayName`

The localized version of the file name. This is the localized version of the LaunchServices call `LSCopyDisplayNameForURL()`/`LSCopyDisplayNameForRef()`. A CFString.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemFSContentChangeDate`

The date the file contents last changed. A CFDate.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemFSCreationDate`

The date and time that the file was created. A CFDate.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemFSExists`

This attribute is deprecated and was never implemented.

Deprecated in Mac OS X v10.4.

Declared in `MDItem.h`.

`kMDItemFSInvisible`

Indicates whether the file is invisible. A CFBoolean.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemFSIsExtensionHidden`

Indicates whether the file extension of the file is hidden. A CFBoolean.

Available in Mac OS X v10.4 and later.

Declared in `MDItem.h`.

`kMDItemFSIsReadable`

This attribute is deprecated and was never implemented.

Deprecated in Mac OS X v10.4.

Declared in `MDItem.h`.

`kMDItemFSIsWriteable`
: This attribute is deprecated and was never implemented.

  Deprecated in Mac OS X v10.4.

  Declared in `MDItem.h`.

`kMDItemFSLabel`
: Index of the Finder label of the file. Possible values are 0 through 7. A CFNumber.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemFSName`
: The file name of the item. A CFString.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemFSNodeCount`
: Number of files in a directory. A CFNumber.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemFSOwnerGroupID`
: The group ID of the owner of the file. A CFNumber.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemFSOwnerUserID`
: The user ID of the owner of the file. A CFNumber.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemFSSize`
: The size, in bytes, of the file on disk. A CFNumber.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

`kMDItemPath`
: The complete path to the file. A CFString.

  Available in Mac OS X v10.4 and later.

  Declared in `MDItem.h`.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`MDItem.h`

MDQuery Reference

Derived From:	CFType
Framework:	CoreServices/CoreServices.h
Declared in	MDQuery.h
Companion guides	Spotlight Overview Spotlight Query Programming Guide

Overview

MDQuery is a CF-compliant object, follows the CF conventions, and can be used with the CF polymorphic functions, such as `CFRetain`. MDQuery encapsulates queries against the System store of the file metadata.

An MDQuery normally executes asynchronously and posts progress notifications as the results are collected. During the gathering phase the query results conform to the specified value lists and sorting.

MDQuery gathers results and processes updates only while the current thread's run loop is running.

For functions that take an MDQueryRef parameter, if this parameter is not a valid MDQueryRef, the behavior is undefined. `NULL` is not a valid MDQueryRef.

For functions that take CF*Ref parameters, such as CFStringRef and CFArrayRef, if this parameter is not a valid CF object of the correct type, the behavior is undefined. `NULL` is not a valid CF*Ref.

Functions by Task

Creating Queries

`MDQueryCreate` (page 155)
Creates a new query instance.

`MDQueryCreateSubset` (page 155)
Creates a new query that is a subset of the specified parentquery.

`MDQuerySetSearchScope` (page 164)
Sets the search scope for a query instance.

Getting and Setting Query Parameters

`MDQueryGetBatchingParameters` (page 158)
Returns the current parameters that control the batching of progress notifications.

`MDQueryCopyValueListAttributes` (page 154)
Returns the list of attribute names for which values are being collected by the query.

`MDQueryCopySortingAttributes` (page 153)
Returns the list of attribute names used to sort the results.

`MDQueryCopyQueryString` (page 153)
Returns the query string of the query.

`MDQuerySetBatchingParameters` (page 161) Deprecated in Mac OS X v10.2
Set the query batching parameters.

Setting Callback Functions

`MDQuerySetCreateResultFunction` (page 162)
Sets the function used to create the result objects of the MDQuery.

`MDQuerySetSortComparator` (page 165)
Sets the function used to sort the results of an MDQuery.

`MDQuerySetCreateValueFunction` (page 163)
Sets the function used to create the value objects of the MDQuery.

Starting, Stopping and Pausing Queries

`MDQueryExecute` (page 157)
Run the query, and populate the query with the results.

`MDQueryStop` (page 165)
Stops the query from generating more results.

`MDQueryDisableUpdates` (page 156)
Disables updates to the query result list.

`MDQueryEnableUpdates` (page 156)
Enables updates to the query result list.

`MDQueryIsGatheringComplete` (page 161)
Returns true if the first phase of a query, the initial result gathering, has finished.

Getting Query Result Values

`MDQueryCopyValuesOfAttribute` (page 154)
Returns the list of values from the results of the query for the specified attribute.

`MDQueryGetAttributeValueOfResultAtIndex` (page 158)
Returns the value of the named attribute for the result at the given index.

`MDQueryGetCountOfResultsWithAttributeValue` (page 158)
Returns the number of results which have the given attribute and attribute value.

`MDQueryGetIndexOfResult` (page 159)
Returns the current index of the given result.

`MDQueryGetResultAtIndex` (page 160)
Returns the current result at the given index.

`MDQueryGetResultCount` (page 160)
Returns the number of results currently collected by the query.

Getting the Type Identifier

`MDQueryGetTypeID` (page 161)
Returns the type identifier of all MDQuery instances

Functions

MDQueryCopyQueryString

Returns the query string of the query.

```
CFStringRef MDQueryCopyQueryString (
   MDQueryRef query
);
```

Parameters

query
The query.

Return Value
A CFStringRef containing the query string.

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

Declared In
`MDQuery.h`

MDQueryCopySortingAttributes

Returns the list of attribute names used to sort the results.

```
CFArrayRef MDQueryCopySortingAttributes (
   MDQueryRef query
);
```

Parameters

query
The query.

Return Value
A CFArrayRef containing the attribute names used to sort the query results.

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

Declared In
`MDQuery.h`

MDQueryCopyValueListAttributes

Returns the list of attribute names for which values are being collected by the query.

```
CFArrayRef MDQueryCopyValueListAttributes (
   MDQueryRef query
);
```

Parameters

query
The query.

Return Value
A CFArrayRef containing the attribute names of the collected values.

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

Declared In
`MDQuery.h`

MDQueryCopyValuesOfAttribute

Returns the list of values from the results of the query for the specified attribute.

```
CFArrayRef MDQueryCopyValuesOfAttribute (
   MDQueryRef query,
   CFStringRef name
);
```

Parameters

query
The query.

name
The attribute name to return the value of. If the attribute is not one of those requested when the query was created the behavior is undefined

Return Value
A CFArrayRef containing the value objects for the specified attribute. The array contents are not ordered and contain only one occurrence of each value. The array contents may change over time if the query is configured for live-updates.

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

Declared In
`MDQuery.h`

MDQueryCreate

Creates a new query instance.

```
MDQueryRef MDQueryCreate (
   CFAllocatorRef allocator,
   CFStringRef queryString,
   CFArrayRef valueListAttrs,
   CFArrayRef sortingAttrs
);
```

Parameters

allocator

The CFAllocator object to be used to allocate memory for the new object. Pass `NULL` or `kCFAllocatorDefault` to use the current default allocator.

queryString

The query expression string for this query.

valueListAttrs

An optional array of attribute names. The query will collect the values of these attributes into uniqued lists that can be used to summarize the results of the query and allow the user to further qualify the search. This parameter may be `NULL` if no value lists are required. Value list collection increases CPU usage and significantly increases the memory usage of an MDQuery. The attribute names are CFStrings.

sortingAttrs

A n array of attribute names used to sort the results, or `NULL` if no sorting is required. The first name in the array is used as the primary sort key, the second as the secondary key, and so on. The comparison of like-typed values is a simple, literal comparison. Sorting increases memory usage and significantly increases the CPU usage of an MDQuery. It is usually more efficent to allow the MDQuery to sort the results than retrieiving the values and sorting the results yourself. The attribute names are CFStrings.

Return Value

An MDQueryRef, or `NULL` on failure. If the query string is empty or malformed the function returns NULL.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`MDQuery.h`

MDQueryCreateSubset

Creates a new query that is a subset of the specified parentquery.

```
MDQueryRef MDQueryCreateSubset (
   CFAllocatorRef allocator,
   MDQueryRef query,
   CFStringRef queryString,
   CFArrayRef valueListAttrs,
   CFArrayRef sortingAttrs
);
```

Parameters

allocator

The CFAllocator object to be used to allocate memory for the new object. Pass NULL or `kCFAllocatorDefault` to use the current default allocator.

query
The parent query

queryString
The query expression string for this query.

valueListAttrs
An optional array of attribute names. The query will collect the values of these attributes into uniqued lists that can be used to summarize the results of the query and allow the user to further qualify the search. This parameter may be `NULL` if no value lists are required. Value list collection increases CPU usage and significantly increases the memory usage of an MDQuery. The attribute names are CFStrings.

sortingAttrs
A n array of attribute names used to sort the results, or `NULL` if no sorting is required. The first name in the array is used as the primary sort key, the second as the secondary key, and so on. The comparison of like-typed values is a simple, literal comparison. Sorting increases memory usage and significantly increases the CPU usage of an MDQuery. It is usually more efficent to allow the MDQuery to sort the results than retrieiving the values and sorting the results yourself. The attribute names are CFStrings.

Return Value
An MDQueryRef, or `NULL` on failure. If the query string is empty or malformed the function returns NULL.

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

Declared In
`MDQuery.h`

MDQueryDisableUpdates

Disables updates to the query result list.

```
void MDQueryDisableUpdates (
   MDQueryRef query
);
```

Parameters

query
The query.

Discussion
This function should be called before iterating over query results that could change due to live-updates. The disabled state is a counter and disabling can be done recursively and from different threads.

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

Declared In
`MDQuery.h`

MDQueryEnableUpdates

Enables updates to the query result list.

```
void MDQueryEnableUpdates (
   MDQueryRef query
);
```

Parameters

query

The query.

Discussion
This function should be called when finished iterating through the list of results. Live-updates to the query results will continue when all the disables have been matched by a corresponding enable.

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

Declared In
`MDQuery.h`

MDQueryExecute

Run the query, and populate the query with the results.

```
Boolean MDQueryExecute (
   MDQueryRef query,
   CFOptionFlags optionFlags
);
```

Parameters

query

The query to execute.

optionFlags

A bitwise OR of the `MDQueryOptionFlags` to be used by the query.

Return Value
Returns `TRUE` if the query was started, `FALSE` otherwise. Queries cannot be executed more than once.

Discussion
Queries only gather results or process updates while the current thread's run loop is running.

Queries have two phases: the initial gathering phase that collects all currently matching results and a second live-update phase. Updates occur during the live-update phase if a change in a file occurs such that it no longer matches the query or if it begins to match the query. Files which begin to match the query are added to the result list, and files which no longer match the query expression are removed from the result list.

Query notifications are posted within the context of the same thread which executes the query.

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

Declared In
`MDQuery.h`

MDQueryGetAttributeValueOfResultAtIndex

Returns the value of the named attribute for the result at the given index.

```
void * MDQueryGetAttributeValueOfResultAtIndex (
   MDQueryRef query,
   CFStringRef name,
   CFIndex idx
);
```

Parameters

query

The query.

name

The attribute name to return the values of. If the attribute is not one of those requested in the *valueListAttrs* or *sortingAttrs* parameters to one of the query creation functions, the result will be `NULL`.

idx

The index into the query's result list. If the index is negative or is equal to or larger than the current number of results in the query, the behavior is undefined.

Return Value

The value of the attribute, or `NULL` if the attribute doesn't exist for the specidied result.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`MDQuery.h`

MDQueryGetBatchingParameters

Returns the current parameters that control the batching of progress notifications.

```
MDQueryBatchingParams MDQueryGetBatchingParameters (
   MDQueryRef query
);
```

Parameters

query

The query.

Return Value

An MDQueryBatchingParams structure with the current batching parameters.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`MDQuery.h`

MDQueryGetCountOfResultsWithAttributeValue

Returns the number of results which have the given attribute and attribute value.

```
CFIndex MDQueryGetCountOfResultsWithAttributeValue (
   MDQueryRef query,
   CFStringRef name,
   CFTypeRef value
);
```

Parameters

query

The query.

name

The attribute name to return the result count of. If the attribute is not one of those requested in the *valueListAttrs* parameter, the behavior is undefined.

value

The attribute value for which to return the number of results with that value. This parameter may be `NULL`, in which case the number of results that do not contain the specified attribute is returned.

Return Value
The number of results containing that attribute and value.

Discussion
This count may change over time if the query allows live-updates.

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

Declared In
`MDQuery.h`

MDQueryGetIndexOfResult

Returns the current index of the given result.

```
CFIndex MDQueryGetIndexOfResult (
   MDQueryRef query,
   const void *result
);
```

Parameters

query

The query.

result

The result object to search for. If a custom create-result function has been set and this parameter is not a valid result object that the provided callbacks can handle, the behavior is undefined. If a custom create-result function has not been set this parameter must be a valid MDItemRef.

Return Value
The index of the given result, or `kCFNotFound` if the value is not one of the query's existing results. If you provided a custom result creation function result, the result will be objects created by that function.

Discussion
If a result-create function has been set, and the equal callback is non-`NULL`, it will be used to test the query's results against the candidate result.

Note that the index of a result can change over time if the query allows live-updates.

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

Declared In
`MDQuery.h`

MDQueryGetResultAtIndex

Returns the current result at the given index.

```
const void * MDQueryGetResultAtIndex (
   MDQueryRef query,
   CFIndex idx
);
```

Parameters

query

The query.

idx

The index into the query's result list. If the index is negative, or is equal to or larger than the current number of results in the query, the behavior is undefined.

Return Value
Returns the MDItemRef currently at the given index, or if a result-creation function has been set, returns the result returned by that function.

Discussion
This function causes the result object to be created if it hasn't been created already. For performance reasons you should only request objects that you require. If possible, call this function to fetch only the results you need to display or otherwise process.

Note that the index of a particular result can change over time if the query is configured to allow live-updates.

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

Declared In
`MDQuery.h`

MDQueryGetResultCount

Returns the number of results currently collected by the query.

```
CFIndex MDQueryGetResultCount (
   MDQueryRef query
);
```

Parameters

query

The query.

Return Value
The number of results in the query.

Discussion
Note that the number of results in a query will change over time as the query's result list is updated.

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

Declared In
`MDQuery.h`

MDQueryGetTypeID

Returns the type identifier of all MDQuery instances

```
CFTypeID MDQueryGetTypeID (
   void
);
```

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

Declared In
`MDQuery.h`

MDQueryIsGatheringComplete

Returns true if the first phase of a query, the initial result gathering, has finished.

```
Boolean MDQueryIsGatheringComplete (
   MDQueryRef query
);
```

Parameters

query
The query.

Return Value
Returns `TRUE` if the first phase of a query has completed, otherwise `FALSE`.

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

Declared In
`MDQuery.h`

MDQuerySetBatchingParameters

Set the query batching parameters.

```
void MDQuerySetBatchingParameters (
   MDQueryRef query,
   MDQueryBatchingParams params
);
```

Parameters

query

The query.

params

An MDQueryBatchingParams structure with the batching parameters to set.

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

Declared In
`MDQuery.h`

MDQuerySetCreateResultFunction

Sets the function used to create the result objects of the MDQuery.

```
void MDQuerySetCreateResultFunction (
   MDQueryRef query,
   MDQueryCreateResultFunction func,
   void *context,
   const CFArrayCallBacks *cb
);
```

Parameters

query

The query.

func

The callback function the MDQuery will use to create its results, such as those returned by the function `MDQueryGetResultAtIndex`. This parameter may be `NULL`, in which case any previous result creation settings are cancelled and the MDQuery will subsequently produce MDItemRefs. If a function is specified and is not of type `MDQueryCreateResultFunction` or does not behave as a `MDQueryCreateResultFunction` must, the behavior is undefined.

context

A pointer-sized user-defined value, that is passed as the third parameter to the create function. MDQuery does not use this value, does not retain the context in any way, and requires that the context be valid for the lifetime of the query. If the context is not what is expected by the create function, the behavior is undefined.

cb

A pointer to a `CFArrayCallBacks` structure initialized with the callbacks for the query to use to manage the created result objects. A copy of the contents of the callbacks structure is made, so that a pointer to a structure on the stack can be passed in, or can be reused for multiple query creations. Only version 0 of the `CFArrayCallBacks` is supported. The retain field may be `NULL`, in which case the MDQuery will not add a retain to the created results for the query. The release field may be `NULL`, in which case the MDQuery will not remove the query's retain (such as the one it gets from the create function) on the result objects when the query is destroyed. If the *`copyDescription`* field is `NULL`, the query will create a simple description for the result objects. If the *`equal`* field is `NULL`, the query will use pointer equality to test for equality of results. This callbacks parameter itself may be `NULL` in which case it is treated as a valid version 0 structure with all fields NULL. Otherwise, if any of the fields are not valid pointers to functions of the correct type, or this parameter is not a valid pointer to a `CFArrayCallBacks` callbacks structure, the behavior is undefined. If any of the value values returned from the create function is not one understood by one or more of the callback functions, the behavior when those callback functions are used is undefined. For example, if the create function can return `NULL`, then `NULL` must be understood by the callback functions as a possible parameter. The retain and release callbacks must be a matched set, you should not assume that the retain function will be unused or that additional reference counts will not be taken on the created results.

Discussion
If no create function is specified for an MDQuery, the default result objects are MDItemRefs. Results created after the function `MDQuerySetCreateResultFunction` is called are created through the specified create function, but values created before the function was set, or after it is unset, are not modified. It is not advisable to change this function after the function `MDQueryExecute` has been called. The create-result function is called lazily as results are requested from a query, it is not called on all results, and may not be called at all. This avoids the cost of creating potentially hundreds of thousands of what might be temporary objects.

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

Declared In
`MDQuery.h`

MDQuerySetCreateValueFunction

Sets the function used to create the value objects of the MDQuery.

```
void MDQuerySetCreateValueFunction (
   MDQueryRef query,
   MDQueryCreateValueFunction func,
   void *context,
   const CFArrayCallBacks *cb
);
```

Parameters

query

The query.

func

The callback function the MDQuery should use to create the value list values, such as those returned by the function `MDQueryCopyValuesOfAttribute`. This parameter may be `NULL`, in which case any previous value creation settings are cancelled and the MDQuery will subsequently produce the default CFTypeRefs. If a function is specified and is not of type `MDQueryCreateValueFunction` or does not behave as a `MDQueryCreateValueFunction` must, the behavior is undefined.

context

A pointer-sized user-defined value, that is passed as the third parameter to the create function. MDQuery does not use this value, does not retain the context in any way, and requires that the context be valid for the lifetime of the query. If the context is not what is expected by the create function, the behavior is undefined.

cb

A pointer to a CFArrayCallBacks structure initialized with the callbacks for the query to use to manage the created value objects. A copy of the contents of the callbacks structure is made, so that a pointer to a structure on the stack can be passed in, or can be reused for multiple query creations. Only version 0 of the `CFArrayCallBacks` is supported. The retain field may be `NULL`, in which case the MDQuery will not add a retain to the created values. The release field may be `NULL`, in which case the MDQuery will do nothing to remove the query's retain (such as the one it gets from the create function) on the value objects when the query is destroyed. If the *copyDescription* field is `NULL`, the query will create a simple description for the value objects. If the *equal* field is `NULL`, the query will use pointer equality to test for equality of values. This callbacks parameter itself may be `NULL` in which case it is treated as a valid version 0 structure with all fields `NULL`. Otherwise, if any of the fields are not valid pointers to functions of the correct type, or this parameter is not a valid pointer to a `CFArrayCallBacks` callbacks structure, the behavior is undefined. If any of the value values returned from the create function is not one understood by one or more of the callback functions, the behavior when those callback functions are used is undefined. For example, if the create function can return `NULL`, then `NULL` must be understood by the callback functions as a possible parameter. The retain and release callbacks must be a matched set, you should not assume that the retain function will be unused or that additional reference counts will not be taken on the created results.

Discussion
Values created after a create function is set will be created using the newly specified function, but existing values are not modified. It is not advisable to change this function after `MDQueryExecute` has been called with the query.

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

Declared In
`MDQuery.h`

MDQuerySetSearchScope

Sets the search scope for a query instance.

```
void MDQuerySetSearchScope (
   MDQueryRef query,
   CFArrayRef scopeDirectories,
   OptionBits scopeOptions
);
```

Parameters

query

The query object to modify.

scopeDirectories

A CFArray of CFStringRef or CFURLRef objects which specify where to search. For convenience the `kMDQueryScopeHome`, `kMDQueryScopeComputer` and `kMDQueryScopeNetwork` constants may also be included in the array.

scopeOptions
Additional options for modifying the search. Currently you must pass 0.

Discussion

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

Declared In
MDQuery.h

MDQuerySetSortComparator

Sets the function used to sort the results of an MDQuery.

```
void MDQuerySetSortComparator (
   MDQueryRef query,
   MDQuerySortComparatorFunction comparator,
   void *context
);
```

Parameters

query
The query.

comparator
The callback function the MDQuery uses to sort the results list. This parameter may be `NULL` which cancels previous sort comparator settings. If a function is specified and is not of type `MDQuerySortComparatorFunction` or does not behave as a `MDQuerySortComparatorFunction` must, the behavior is undefined.

context
A pointer-sized user-defined value, that is passed as the third parameter to the create function. MDQuery does not use this value, does not retain the context in any way, and requires that the context be valid for the lifetime of the query. If the context is not what is expected by the create function, the behavior is undefined.

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

Declared In
MDQuery.h

MDQueryStop

Stops the query from generating more results.

```
void MDQueryStop (
   MDQueryRef query
);
```

Parameters

query
The query.

Discussion
Queries may be executed only once and cannot be restarted. The query will first complete processing any unprocessed results.do. That may trigger a progress notification, so be aware of that if you are stopping a query from within your progress note handler; that is, during this function, a recursive progress and/or finished notification might occur, which might recursively call your notification handler. It is safe to call this function recursively. You would call this function to stop a query that is generating way too many results to be useful, but still want to access the results that have come in so far. If a query is stopped before the gathering phase finishes, it will not report itself as finished, nor will it send out a finished notification.

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

Declared In
`MDQuery.h`

Callbacks

MDQueryCreateResultFunction

Callback function used to create the result objects stored and returned by a query.

```
typedef const void * (*MDQueryCreateResultFunction) (
 MDQueryRef query,
 MDItemRef item,
 void *context
);

forthcoming
```

Parameters

query
The query instance.

item
The default MDItemRef for the result.

context
The user-defined context parameter provided to the `MDQuerySetCreateResultFunction` function.

Return Value
The function must return a pointer-sized value that can be managed with the callbacks which were set at the same time the create function was given to the query. The value must be returned with a reference (such as if the retain callback had been called on it), as implied by the Create name. If this function doesn't wish to create a new object it can return the given MDItemRef, but must also return it with a new retain, and the callbacks must be able to handle an MDItemRef as an input value. If this function returns NULL, NULL will be stored for the moment in the query, MDQueryGetResultAtIndex() may return NULL for that result, and the next time the query wants the result, it will call this function again.

Discussion
The function may hold onto the given attribute name and/or value in some other data structure, but must retain them for them to remain valid.

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

Declared In
`MDQuery.h`

MDQueryCreateValueFunction

Callback function used to create the value objects stored and returned by a query.

```
typedef const void * (*MDQueryCreateValueFunction) (
 MDQueryRef query,
 CFStringRef attrName,
 CFTypeRef attrValue,
 void *context
);

forthcoming
```

Parameters

query
The query instance.

attrName
The attribute name of the value.

attrValue
The default value of the value.

context
The user-defined context parameter provided in the `MDQuerySetCreateValueFunction` function.

Return Value
The function must return a pointer-sized value that can be managed with the callbacks which were set at the same time the create function was given to the query. The value must be returned with a reference (such as if the retain callback had been called on it), as implied by the Create name. If this function doesn't wish to create a new object, it can return the given CFTypeRef, but must also return it with a new retain, and the callbacks must be able to handle a CFTypeRef as an input value.

Discussion
The function may hold onto the given attribute name and/or value in some other data structure, but must retain them for them to remain valid

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

Declared In
`MDQuery.h`

MDQuerySortComparatorFunction

Callback function used to sort the results of a query.

```
typedef CFComparisonResult (*MDQuerySortComparatorFunction) (
 const CFTypeRef attrs1[],
 const CFTypeRef attrs2[],
 void *context
);

asdfasdfasdfasdfadsf
```

Parameters

query

The query instance.

attrs1

A C array of attribute values for a result. The values occur in the array in the same order and position that the attribute names were passed in the `sortingAttrs` array when the query was created. The values of the attributes will be `NULL` if the attribute doesn't exist for a result or if read access to that attribute is not allowed.

attrs2

A C array of attribute values for a result. The values occur in the array in the same order and position that the attribute names were passed in the `sortingAttrs` array when the query was created. The values of the attributes will be `NULL` if the attribute doesn't exist for a result or if read access to that attribute is not allowed.

context

The user-defined context parameter provided in the function `MDQuerySetSortComparator`.

Return Value

The function must return one of the CFComparisonResults `kCFCompareLessThan`, `kCFCompareEqualTo`, or `kCFCompareGreaterThan`. There is no provision for unordered results. The comparison should be a total order relation and produce the same results for the same inputs.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`MDQuery.h`

Data Types

Batching Parameters

MDQueryBatchingParams

Structure containing the progress notification batching parameters of a MDQuery.

```
typedef struct {
    size_t first_max_num;
    size_t first_max_ms;
    size_t progress_max_num;
    size_t progress_max_ms;
    size_t update_max_num;
    size_t update_max_ms;
} MDQueryBatchingParams;
```

Fields

`first_max_num`

The maximum number of results that can accumulate before the first progress notification is sent. This value is used only during the initial result-gathering phase of a query.

`first_max_ms`

The maximum number of milliseconds that can pass before the first progress notification is sent. This value is advisory, in that the notification will be triggered at some point after `first_max_ms` milliseconds have passed since the query began accumulating results. This value is used only during the initial result-gathering phase of a query.

`progress_max_num`

The maximum number of results that can accumulate before additional progress notifications are sent. This value is used only during the initial result-gathering phase of a query.

`progress_max_ms`

The maximum number of milliseconds that can pass before additional progress notifications are sent. This value is advisory, in that the notification will be triggered at some point after `progress_max_ms` milliseconds have passed since the query began accumulating results. This value is used only during the initial result-gathering phase of a query.

`update_max_num`

The maximum number of results that can accumulate before an update notification is sent. This value is used only during the live-update phase of a query.

`update_max_ms`

The maximum number of milliseconds that can pass before an update notification is sent. This value is advisory, in that the notification will be triggered at some point after `update_max_ms` milliseconds have passed since the query began accumulating results. This value is used only during the live-update phase of a query.

Discussion

The default batching parameters are undefined and subject to change.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`MDQuery.h`

Miscellaneous

MDQueryRef

A reference to a MDQuery object.

```
typedef struct __MDQuery *MDQueryRef;
```

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

Declared In
`MDQuery.h`

Constants

Query Option Flags

MDQueryOptionsFlags

Specify the execution mode for a query.

```
typedef enum {
    kMDQuerySynchronous = 1,
    kMDQueryWantsUpdates = 4,
} MDQueryOptionFlags;
```

Constants

`kMDQuerySynchronous`
Specifies that a query should block during the initial gather phase. The query's run loop will run in the default mode. If this option is not specified the query function returns immediately after starting the query asynchronously.

Available in Mac OS X v10.4 and later.

Declared in `MDQuery.h`.

`kMDQueryWantsUpdates`
Specifies that a query should provide live-updates to the result list after the initial gathering phase. Updates occur during the live-update phase if a change in a file occurs such that it no longer matches the query or if it begins to match the query. Files which begin to match the query are added to the result list, and files which no longer match the query expression are removed from the result list. Currently, this option is ignored if the `kMDQuerySynchronous` parameter is specified. This is subject to change, and you should always pass the value appropraite to the required behavior.

Available in Mac OS X v10.4 and later.

Declared in `MDQuery.h`.

Notifications

kMDQueryDidFinishNotification

Indicates that a query has finished with the initial result-gathering phase.

```
const CFStringRef kMDQueryDidFinishNotification;
```

Constants
`kMDQueryDidFinishNotification`
Posted to indicate that the query has finished the initial result-gathering phase.

Available in Mac OS X v10.4 and later.

Declared in `MDQuery.h`.

Discussion
The query results list is not updated as a result of this notification.

This notification is only sent to the application's notification center.

kMDQueryDidUpdateNotification

Indicates that a query's results list has change during the live-update phase of a query.

```
const CFStringRef kMDQueryDidUpdateNotification;
```

Constants
`kMDQueryDidUpdateNotification`
Notification posted to indicate that a change has occured to the query's resuls list during the live-update phase of a query's execution.

Available in Mac OS X v10.4 and later.

Declared in `MDQuery.h`.

Discussion
The info dictionary of the notification can contain `kMDQueryUpdateAddedItems`, `kMDQueryUpdateChangedItems`, and `kMDQueryUpdateRemovedItems` keys.

This notification is only sent to the application's notification center.

kMDQueryProgressNotification

Indicates that a query's results list has change during the initial result-gathering phase of a query.

```
const CFStringRef kMDQueryProgressNotification;
```

Constants
`kMDQueryProgressNotification`
Notification posted to indicate that a change has occurred to the query's results list during the initial result-gathering phase of execution.

Available in Mac OS X v10.4 and later.

Declared in `MDQuery.h`.

Discussion
New items are typically added during this phase, however it is possible for items to be removed or updated, if the original file is changed. The info dictionary of the notification can contain `kMDQueryUpdateChangedItems` and `kMDQueryUpdateRemovedItems` keys.

For performance reasons added results are not indicated in progress notifications, to avoid the cost of creating the result objects.

This notification is only sent to the application's notification center.

Notification Info Keys

Query Result Change Keys

Specify the items that have changed in the query results.

```
const CFStringRef kMDQueryUpdateAddedItems;
const CFStringRef kMDQueryUpdateChangedItems;
const CFStringRef kMDQueryUpdateRemovedItems;
```

Constants

`kMDQueryUpdateAddedItems`

An array that identifies the items that have been added to the query results. This list only contains result objects that have previously been created, result objects that have not been created are not included.

Available in Mac OS X v10.4 and later.

Declared in `MDQuery.h`.

`kMDQueryUpdateChangedItems`

An array that identifies the items that have changed in the query results. This list only contains result objects that have previously been created, result objects that have not been created are not included.

Available in Mac OS X v10.4 and later.

Declared in `MDQuery.h`.

`kMDQueryUpdateRemovedItems`

An array that identifies the items that have been removed from the query results. This list only contains result objects that have previously been created, result objects that have not been created are not included.

Available in Mac OS X v10.4 and later.

Declared in `MDQuery.h`.

Query Search Scope Keys

Specify the scope of a query's search.

```
const CFStringRef kMDQueryScopeHome;
const CFStringRef kMDQueryScopeComputer;
const CFStringRef kMDQueryScopeNetwork;
```

Constants

`kMDQueryScopeHome`

Specifies that the query should be restricted to the volume and directory that contains the current user's home directory.

Available in Mac OS X v10.4 and later.

Declared in `MDQuery.h`.

`kMDQueryScopeComputer`
: Specifies that the query should be restricted to all locally mounted volumes, plus the user's home directory (which may be on a remote volume).

  Available in Mac OS X v10.4 and later.

  Declared in `MDQuery.h`.

`kMDQueryScopeNetwork`
: Specifies that the query should include all user mounted remote volumes.

  Available in Mac OS X v10.4 and later.

  Declared in `MDQuery.h`.

Discussion
These constants can be passed in the *`scopeDirectories`* array to the function `MDQuerySetSearchScope`.

Result Relevance Sorting Key

Key used in a user notification's description dictionary that indicates the relevance of a result.

```
const CFStringRef kMDQueryResultContentRelevance;
```

Constants
`kMDQueryResultContentRelevance`
: A CFNumberRef with a floating point value between 0.0 and 1.0 inclusive.

  Available in Mac OS X v10.4 and later.

  Declared in `MDQuery.h`.

Discussion
The relevance value indicates the relevance of the content of a result object. The relevance is computed based on the value of the result itself, not on its relevance to the other results returned by the query.

The relevance value is for the content of the object only, not on the result item as a whole, and may not be computed if the item matches theq uery through evaluation of other attributes

If the value is not computed it is treated as an attribute on the item that does not exist.

Managers

Alias Manager Reference

Framework: CoreServices/CoreServices.h

Declared in Aliases.h

Overview

The Alias Manager creates and resolves alias records, which are data structures that describe file system objects (files, directories, and volumes.) An alias record contains a "fingerprint" of a file system object. You can store the alias record instead of a file system reference, and use the Alias Manager to find the object again when it's needed. The Alias Manager contains algorithms for locating objects that have been moved, renamed, copied, or restored from backup.

The exact makeup of an alias record depends on the file system in which the object resides. The Alias Manager takes advantage of persistent object ids, creation dates, file types, creator codes and the like if they are available. By default, an object at the location stored in the alias record will be considered a stronger match than an object with the same file id in a different location. (You can alter this behavior by passing flags to the functions that resolve the alias.)

The Alias Manager supports two types of alias records. The standard alias contains as much information as the Alias Manager can gather from the underlying file system. The minimal alias only stores a subset of the information in a standard alias record. A minimal alias may be used when the object is unlikely to move, the reference is to be short-lived, or space is a critical issue (the exact space savings depends on the underlying file system format.) The standard alias is the preferred format because it is more robust.

The Finder supports the creation and use of alias files that contain alias records. Currently, Mac OS X does not provide a way for other applications to create these alias files. The Alias Manager can identify and resolve Finder alias files, but it cannot create them.

Functions by Task

Creating and Updating Alias Records

`FSNewAlias` (page 188)
Creates a new alias record, given a target file or directory.

`FSNewAliasUnicode` (page 191)
Creates a new alias record, given the Unicode name and parent directory of the target.

`FSNewAliasFromPath` (page 189)
Creates a new alias record, given the pathname of the target file or directory.

`FSNewAliasMinimal` (page 190)
Creates a new minimal alias record, given a target file or directory.

`FSNewAliasMinimalUnicode` (page 190)
Creates a minimal alias, given the Unicode name and parent directory of the target.

`FSUpdateAlias` (page 196)
Updates an alias record for a specified target.

Getting Alias Size

`GetAliasSize` (page 198)
Gets the size of an alias record referenced by a handle.

`GetAliasSizeFromPtr` (page 198)
Gets the size of an alias record referenced by a pointer.

Getting and Setting Alias User Types

`GetAliasUserType` (page 198)
Gets the user type for an alias record referenced by a handle.

`SetAliasUserType` (page 212)
Sets the user type for an alias record referenced by a handle.

`GetAliasUserTypeFromPtr` (page 199)
Gets the user type for the alias record referenced by a pointer.

`SetAliasUserTypeWithPtr` (page 212)
Sets the user type for the alias record referenced by a pointer.

Resolving and Reading Alias Records

`FSCopyAliasInfo` (page 181)
Returns information from an alias handle.

`FSMatchAliasBulk` (page 185)
Identifies a list of possible matches for an alias.

`FSResolveAlias` (page 192)
Returns an `FSRef` to the single most likely target of an alias record.

`FSResolveAliasWithMountFlags` (page 195)
Returns an `FSRef` to the target of an alias.

`FSMatchAlias` (page 184) Deprecated in Mac OS X v10.5
Identifies a list of possible matches for an alias. (Deprecated. Use `FSMatchAliasBulk` (page 185) instead.)

`FSMatchAliasNoUI` (page 187) Deprecated in Mac OS X v10.5
Identifies a list of possible matches for an alias without any user interaction. (Deprecated. Use `FSMatchAliasBulk` (page 185) with the `kARMNoUI` flag instead.)

Working With Finder Alias Files

`FSFollowFinderAlias` (page 182)
Resolves an alias record obtained from a Finder alias file.

`FSIsAliasFile` (page 183)
Determines whether a file system object is an alias file, a data file, or a folder.

`FSResolveAliasFile` (page 193)
Resolves an alias contained in an alias file.

`FSResolveAliasFileWithMountFlags` (page 194)
Resolves an alias contained in an alias file.

Working With Universal Procedure Pointers to Alias Manager Callbacks

`NewAliasFilterUPP` (page 204)
Creates a new universal procedure pointer (UPP) to an alias filtering callback function.

`DisposeAliasFilterUPP` (page 180)
Disposes of a universal procedure pointer (UPP) to an alias filtering callback function.

`InvokeAliasFilterUPP` (page 199)
Calls your alias filtering callback function.

Deprecated Functions

Alias Manager functions that use the `FSSpec` data type have been deprecated. Instead, you should use the equivalent `FSRef`–based functions, which include support for features such as unicode and long filenames. For more information on `FSSpec` and `FSRef` types, see *File Manager Reference*.

`GetAliasInfo` (page 197) Deprecated in Mac OS X v10.3
Gets information from an alias record without actually resolving the record. (Deprecated. Use `FSCopyAliasInfo` (page 181) instead.)

`FollowFinderAlias` (page 181) Deprecated in Mac OS X v10.5
Resolves an alias record obtained from a Finder alias file. (Deprecated. Use `FSFollowFinderAlias` (page 182) instead.)

`MatchAliasNoUI` (page 202) Deprecated in Mac OS X v10.5
Identifies a list of possible matches for an alias without any user interaction. (Deprecated. Use `FSMatchAliasBulk` (page 185) with the `kARMNoUI` flag instead.)

`ResolveAliasFileWithMountFlags` (page 209) Deprecated in Mac OS X v10.5
Resolves an alias contained in an alias file. (Deprecated. Use `FSResolveAliasFileWithMountFlags` (page 194) instead.)

`IsAliasFile` (page 200) Deprecated in Mac OS X v10.4
Determines whether a file system object is an alias file, a data file, or a folder. (Deprecated. Use `FSIsAliasFile` (page 183) instead.)

`MatchAlias` (page 201) Deprecated in Mac OS X v10.4
Identifies a list of possible matches for an alias and passes the list through an optional selection filter. The filter can return more than one possible match. (Deprecated. Use `FSMatchAliasBulk` (page 185) instead.)

`NewAlias` (page 203) Deprecated in Mac OS X v10.4
Creates a complete alias record. (Deprecated. Use `FSNewAlias` (page 188) instead.)

`NewAliasMinimal` (page 205) Deprecated in Mac OS X v10.4
Creates a short alias record quickly. (Deprecated. Use `FSNewAliasMinimal` (page 190) instead.)

`NewAliasMinimalFromFullPath` (page 205) Deprecated in Mac OS X v10.4
Creates an alias record that contains only the full pathname of the target. (Deprecated. Use `FSNewAliasMinimal` (page 190) or `FSNewAliasMinimalUnicode` (page 190) instead.)

`ResolveAlias` (page 206) Deprecated in Mac OS X v10.4
Identifies the single most likely target of an alias record. (Deprecated. Use `FSResolveAlias` (page 192) instead.)

`ResolveAliasFile` (page 208) Deprecated in Mac OS X v10.4
Resolves an alias contained in an alias file. (Deprecated. Use `FSResolveAliasFile` (page 193) instead.)

`ResolveAliasFileWithMountFlagsNoUI` (page 210) Deprecated in Mac OS X v10.4
Resolves an alias file without any user interaction. (Deprecated. Use `FSResolveAliasFileWithMountFlags` (page 194) with the `kResolveAliasFileNoUI` flag instead.)

`ResolveAliasWithMountFlags` (page 211) Deprecated in Mac OS X v10.4
Identifies the target of an alias. (Deprecated. Use `FSResolveAliasWithMountFlags` (page 195) instead.)

`UpdateAlias` (page 213) Deprecated in Mac OS X v10.4
Updates an alias record. (Deprecated. Use `FSUpdateAlias` (page 196) instead.)

Functions

DisposeAliasFilterUPP

Disposes of a universal procedure pointer (UPP) to an alias filtering callback function.

```
void DisposeAliasFilterUPP (
   AliasFilterUPP userUPP
);
```

Parameters

userUPP
The UPP to dispose of.

Discussion

See `AliasFilterProcPtr` (page 215) for more information on alias filtering callback functions.

Availability

Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In

`Aliases.h`

FollowFinderAlias

Resolves an alias record obtained from a Finder alias file. (Deprecated in Mac OS X v10.5. Use `FSFollowFinderAlias` (page 182) instead.)

```
OSErr FollowFinderAlias (
   const FSSpec *fromFile,
   AliasHandle alias,
   Boolean logon,
   FSSpec *target,
   Boolean *wasChanged
);
```

Parameters

fromFile

A pointer to a file system specification specifying a file for a first attempt at a relative resolution; pass a pointer to the alias file's `FSSpec` forthis parameter.

alias

A handle to the alias record taken from the alias file's resources.

logon

If `true`, the Alias Manager attempts to mount a volume if necessary to complete the resolution of the alias.

target

A pointer to an `FSSpec` structure. On return, this `FSSpec` refers to the target found by the resolution.

wasChanged

A pointer to a Boolean value. `FollowFinderAlias` sets this value to `true` if it has updated the alias record. If the alias has been updated, you should call `ChangedResource` and `WriteResource` if the updated record should be saved in the resource file.

Return Value

A result code.

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

`Aliases.h`

FSCopyAliasInfo

Returns information from an alias handle.

```
OSStatus FSCopyAliasInfo (
   AliasHandle inAlias,
   HFSUniStr255 *targetName,
   HFSUniStr255 *volumeName,
   CFStringRef *pathString,
   FSAliasInfoBitmap *whichInfo,
   FSAliasInfo *info
);
```

Parameters

inAlias

A handle to the alias record from which to get information.

targetName

A pointer to a string that, on return, contains the name of the target item. Pass `NULL` if you do not want this information returned.

volumeName

A pointer to a string that, on return, contains the name of the volume the target resides on. Pass `NULL` if you do not want this information returned.

pathString

A pointer a CFString that, on return, contains the POSIX path to the target. Pass `NULL` if you do not want this information returned.

whichInfo

A pointer to a variable of type `FSAliasInfoBitmap`. On return, this field indicates which fields in the alias information block, specified in the `info` parameter, contain valid data. See "Alias Information Masks" (page 218) for a description of the values that may be returned here. This parameter may be `NULL`.

info

A pointer to a structure of type `FSAliasInfo` (page 217). On return, this structure contains information about the alias. Pass `NULL` if you do not want this information returned.

Return Value
A result code.

Discussion
This function returns the requested information from the alias handle passed in the *inAlias* parameter. The information is gathered only from the alias record, so it may not match what is on disk. No disk input/output is performed.

The `FSCopyAliasInfo` function adds support for unicode filenames and filenames longer than 32 bytes. It replaces the `GetAliasInfo` function.

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

Declared In
`Aliases.h`

FSFollowFinderAlias

Resolves an alias record obtained from a Finder alias file.

```
OSErr FSFollowFinderAlias (
   FSRef *fromFile,
   AliasHandle alias,
   Boolean logon,
   FSRef *target,
   Boolean *wasChanged
);
```

Parameters

fromFile

A pointer to the file to use for a first attempt at a relative resolution; pass a pointer to the alias file's `FSRef` for this parameter.

alias

A handle to the alias record taken from the alias file's resources.

logon

If `true`, the Alias Manager attempts to mount a volume if necessary to complete the resolution of the alias.

target

A pointer to an `FSRef` structure. On return, this `FSRef` refers to the target found by the resolution.

wasChanged

A pointer to a Boolean value. `FSFollowFinderAlias` sets this value to `true` if it has updated the alias record.

Return Value
A result code.

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

Declared In
`Aliases.h`

FSIsAliasFile

Determines whether a file system object is an alias file, a data file, or a folder.

```
OSErr FSIsAliasFile (
   const FSRef *fileRef,
   Boolean *aliasFileFlag,
   Boolean *folderFlag
);
```

Parameters

fileRef

A pointer to the file system object to test.

aliasFileFlag

A pointer to a Boolean variable. On return, a value of `TRUE` indicates that the object specified in the *fileRef* parameter is an alias file. A value of `FALSE` indicates that the object is not an alias file.

folderFlag

A pointer to a Boolean variable. On return, a value of `TRUE` indicates that the object specified in the *fileRef* parameter is a folder. A value of `FALSE` indicates that the object is a file.

Return Value
A result code.

Discussion
Table 10-1 summarizes the information that this function provides about the object specified in the `fileRef` parameter:

Table 10-1 Information about a file system object

Alias flag	Folder flag	Object kind
T	F	Alias file
F	F	Data file
F	T	Folder

Note that if `fileRef` is an alias file, this function does not provide any information about the object to which the alias refers. To find out whether this object is a file or a folder, you can use `FSResolveAliasFile` (page 193).

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

Related Sample Code
QTCarbonShell

Declared In
`Aliases.h`

FSMatchAlias

Identifies a list of possible matches for an alias. (Deprecated in Mac OS X v10.5. Use `FSMatchAliasBulk` (page 185) instead.)

```
OSErr FSMatchAlias (
   const FSRef *fromFile,
   unsigned long rulesMask,
   AliasHandle inAlias,
   short *aliasCount,
   FSRef *aliasList,
   Boolean *needsUpdate,
   AliasFilterUPP aliasFilter,
   void *yourDataPtr
);
```

Parameters

fromFile

A pointer to the starting point for a relative search. You may pass `NULL` if you do not want this function to perform a relative search.

rulesMask

A set of rules to guide the resolution. Pass the sum of all of the rules you want to invoke. For a description of the values you can use in this parameter, see "Matching Constants" (page 220).

inAlias
A handle to the alias record to be resolved.

aliasCount
On input, a pointer to the maximum number of possible matches to return. On output, the actual number of matches returned.

aliasList
A pointer to an array of FSRef structures. On return, this array holds the results of the search, a list of possible candidates.

needsUpdate
A pointer to a Boolean flag that, on return, indicates whether the alias record needs to be updated.

aliasFilter
An application-defined filter function. The Alias Manager executes this function each time it identifies a possible match. Your filter function returns a Boolean value that determines whether the possible match is discarded (true) or added to the list of possible targets (false). It can also terminate the search by setting the variable parameter quitFlag. See AliasFilterProcPtr (page 215) for a description of the filter function.

yourDataPtr
A pointer to data to be passed to the filter function. The yourDataPtr parameter can point to any data your application might need in the filter function.

Return Value
A result code. When it finds the specified volume and parent directory but fails to find the target file or directory in that location, FSMatchAlias returns fnfErr. Note that the file system objects in the aliasList parameter are not valid in this case.

Discussion
After it identifies a target, FSMatchAlias compares some key information about the target with the same information in the record. If the information does not match, FSMatchAlias sets the *needsUpdate* flag to true.

The FSMatchAlias function also sets the *needsUpdate* flag to true if it identifies a list of possible matches rather than a single match or if kARMsearchRelFirst is set in the *rulesMask* parameter but the target is identified through either an absolute search or an exhaustive search. Otherwise, the FSMatchAlias function sets the *needsUpdate* flag to false. FSMatchAlias always sets the *needsUpdate* flag to false when resolving an alias created by FSNewAliasMinimal. If you want to update the alias record to reflect the final results of the resolution, call FSUpdateAlias.

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
Aliases.h

FSMatchAliasBulk

Identifies a list of possible matches for an alias.

```
OSStatus FSMatchAliasBulk (
    const FSRef *fromFile,
    unsigned long rulesMask,
    AliasHandle inAlias,
    short *aliasCount,
    FSRef *aliasList,
    Boolean *needsUpdate,
    FSAliasFilterProcPtr aliasFilter,
    void *yourDataPtr
);
```

Parameters

fromFile

A pointer to the starting point for a relative search. You may pass `NULL` if you do not want this function to perform a relative search. By default, this function performs a relative search only if the absolute search does not find a match. If you want to perform the relative search first, you should pass `kARMSearchRelFirst` in the `rulesMask` parameter.

rulesMask

A set of rules to guide the resolution. Pass the sum of all of the rules you want to invoke. For a description of the values you can use in this parameter, see "Matching Constants" (page 220).

inAlias

A handle to the alias record to be resolved.

aliasCount

On input, a pointer to the maximum number of possible matches to return. On output, the actual number of matches returned.

aliasList

A pointer to an array of `FSRef` structures. On output, this array holds the results of the search, a list of possible candidates.

needsUpdate

A pointer to a Boolean flag that, on output, indicates whether the alias record needs to be updated. For more information about this parameter, see the Discussion.

aliasFilter

An optional application-defined filter function. The Alias Manager calls your filter function each time it identifies a possible match or after the search has continued for three seconds without a match. Your filter function returns a Boolean value that determines whether the possible match is discarded (`true`) or added to the list of possible targets (`false`). It can also terminate the search by setting the variable parameter `quitFlag`. See `FSAliasFilterProcPtr` (page 215) for a description of the filter function.

yourDataPtr

A pointer to data to be passed to the filter function, or `NULL`. The `yourDataPtr` parameter can point to any data your application might need in the filter function.

Return Value

A result code. If the Alias Manager finds the specified volume and parent directory but fails to find the target file or directory in that location, the return value is `fnfErr` and the elements in the `aliasList` parameter are not valid.

Discussion

After it identifies a target, this function compares some key information about the target with the same information in the record. If the information does not match, this function sets the *needsUpdate* flag to `true`. This function also sets the *needsUpdate* flag to `true` if it identifies a list of possible matches rather than a single match or if `kARMsearchRelFirst` is set in the *rulesMask* parameter but the target is identified

through either an absolute search or an exhaustive search. Otherwise, this function sets the *needsUpdate* flag to false. This function always sets the *needsUpdate* flag to false when resolving an alias created by FSNewAliasMinimal. To update the alias record to reflect the final results of the resolution, use the function FSUpdateAlias (page 196).

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

Declared In
Aliases.h

FSMatchAliasNoUI

Identifies a list of possible matches for an alias without any user interaction. (Deprecated in Mac OS X v10.5. Use FSMatchAliasBulk (page 185) with the kARMNoUI flag instead.)

```
OSErr FSMatchAliasNoUI (
   const FSRef *fromFile,
   unsigned long rulesMask,
   AliasHandle inAlias,
   short *aliasCount,
   FSRef *aliasList,
   Boolean *needsUpdate,
   AliasFilterUPP aliasFilter,
   void *yourDataPtr
);
```

Parameters

fromFile
A pointer to the starting point for a relative search. You may pass NULL if you do not want this function to perform a relative search.

rulesMask
A set of rules to guide the resolution. Pass the sum of all of the rules you want to invoke. For a description of the values you can use in this parameter, see "Matching Constants" (page 220).

inAlias
A handle to the alias record to be resolved.

aliasCount
On input, a pointer to the maximum number of possible matches to return. On output, the actual number of matches returned.

aliasList
A pointer to an array of FSRef structures. On return, this array holds the results of the search, a list of possible candidates.

needsUpdate
A pointer to a Boolean flag that, on return, indicates whether the alias record needs to be updated.

aliasFilter
An application-defined filter function. The Alias Manager executes this function each time it identifies a possible match. Your filter function returns a Boolean value that determines whether the possible match is discarded (true) or added to the list of possible targets (false). It can also terminate the search by setting the variable parameter quitFlag. See AliasFilterProcPtr (page 215) for a description of the filter function.

yourDataPtr

A pointer to data to be passed to the filter function. The `yourDataPtr` parameter can point to any data your application might need in the filter function.

Return Value
A result code.

Discussion
The `FSMatchAliasNoUI` function operates in much the same way as the `FSMatchAlias` function; however, it does not present an interface to the user. Additionally, the `FSMatchAliasNoUI` function does not mount network volumes, even when it is possible to mount the volume without user interaction. See the discussion of `FSMatchAlias` (page 184) for more information.

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
`Aliases.h`

FSNewAlias

Creates a new alias record, given a target file or directory.

```
OSErr FSNewAlias (
   const FSRef *fromFile,
   const FSRef *target,
   AliasHandle *inAlias
);
```

Parameters

fromFile

A pointer to the starting point for a relative search. You may pass `NULL` if you do not need relative search information in the alias record. The files or directories specified in the *fromFile* and *target* parameters must reside on the same volume.

target

A pointer to the target file or directory of the alias.

inAlias

A pointer to an alias handle. On return, this handle refers to the newly created alias record. If the function fails to create an alias record, it sets *inAlias* to `NULL`.

Return Value
A result code. If the specified target is valid, this function creates an alias record for the target and returns `noErr`. Any other return value indicates that this function did not create an alias record.

Discussion
The `FSNewAlias` function creates an alias record that describes the specified target. It allocates the storage, fills in the record, and puts a record handle to that storage in the *inAlias* parameter. `FSNewAlias` records the full pathname of the target and a collection of other information relevant to locating the target, verifying the target, and mounting the target's volume, if necessary. You can have `FSNewAlias` store relative search information as well by supplying a starting point for a relative search.

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

Related Sample Code
QTCarbonShell

Declared In
`Aliases.h`

FSNewAliasFromPath

Creates a new alias record, given the pathname of the target file or directory.

```
OSErr FSNewAliasFromPath (
   const char *fromFilePath,
   const char *targetPath,
   OptionBits flags,
   AliasHandle *inAlias,
   Boolean *isDirectory
);
```

Parameters

fromFilePath
A C string that specifies the starting point for a relative search. The string should contain a UTF-8 pathname. You may pass `NULL` if you do not need relative search information in the alias record.

targetPath
A C string that contains the full UTF-8 pathname of the target object.

flags
Reserved for future use. Currently, you should pass 0.

inAlias
A pointer to an alias handle. On output, this handle refers to the newly created alias record.

isDirectory
A pointer to a Boolean value. On input, if the target does not exist, set the value to `true` if the target is a directory or `false` if it is not. (Pass `NULL` if you are not sure whether the target is a directory.) On output, if the target exists, the value is `true` if the target is a directory, `false` if it is not.

Return Value
A result code. For more information, see the Discussion.

Discussion
If the specified target exists, this function creates an alias record for the target and returns `noErr`. If the parent directory specified in the target pathname exists but the target itself does not exist, this function creates an alias record for the target and returns `fnfErr`. Any other return value indicates that this function did not create an alias record.

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

Declared In
`Aliases.h`

FSNewAliasMinimal

Creates a new minimal alias record, given a target file or directory.

```
OSErr FSNewAliasMinimal (
   const FSRef *target,
   AliasHandle *inAlias
);
```

Parameters

target

A pointer to the target of the alias record.

inAlias

A pointer to an alias handle. On return, this handle refers to the newly created alias record. If the function fails to create an alias record, it sets `inAlias` to `NULL`.

Return Value

A result code. If the specified target is valid, this function creates an alias record for the target and returns `noErr`. Any other return value indicates that this function did not create an alias record.

Discussion

The `FSNewAliasMinimal` function creates an alias record that contains only the minimum information necessary to describe the target. The `FSNewAliasMinimal` function uses the standard alias record data structure, but it fills in only parts of the record.

The `FSResolveAlias` (page 192) function never updates a minimal alias record.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Aliases.h`

FSNewAliasMinimalUnicode

Creates a minimal alias, given the Unicode name and parent directory of the target.

```
OSErr FSNewAliasMinimalUnicode (
   const FSRef *targetParentRef,
   UniCharCount targetNameLength,
   const UniChar *targetName,
   AliasHandle *inAlias,
   Boolean *isDirectory
);
```

Parameters

targetParentRef

A pointer to the parent directory of the target.

targetNameLength

The number of Unicode characters in the target's name.

targetName

A pointer to the Unicode name of the target.

inAlias

A pointer to an alias handle. On return, this handle refers to the newly created alias record.

isDirectory
A pointer to a Boolean value. On input, if the target does not exist, set the value to `true` if the target is a directory or `false` if it is not. (Pass `NULL` if you are not sure whether the target is a directory.) On output, if the target exists, the value is `true` if the target is a directory, `false` if it is not.

Return Value
A result code. For more information, see the Discussion.

Discussion
If the specified target exists, this function creates an alias record for the target and returns `noErr`. If the parent directory exists but the target itself does not exist, this function creates an alias record for the target and returns `fnfErr`. Any other return value indicates that this function did not create an alias record.

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

Declared In
`Aliases.h`

FSNewAliasUnicode

Creates a new alias record, given the Unicode name and parent directory of the target.

```
OSErr FSNewAliasUnicode (
    const FSRef *fromFile,
    const FSRef *targetParentRef,
    UniCharCount targetNameLength,
    const UniChar *targetName,
    AliasHandle *inAlias,
    Boolean *isDirectory
);
```

Parameters

fromFile
A pointer to the starting point for a relative search. You may pass `NULL` if you do not need relative search information in the alias record.

targetParentRef
A pointer to the parent directory of the target.

targetNameLength
The number of Unicode characters in the target's name.

targetName
A pointer to the Unicode name of the target.

inAlias
A pointer to an alias handle. On return, this handler refers to the newly created alias record.

isDirectory
A pointer to a Boolean value. On input, if the target does not exist, set the value to `true` if the target is a directory or `false` if it is not. (Pass `NULL` if you are not sure whether the target is a directory.) On output, if the target exists, the value is `true` if the target is a directory, `false` if it is not.

Return Value
A result code. For more information, see the Discussion.

Discussion
If the specified target exists, this function creates an alias record for the target and returns `noErr`. If the parent directory exists but the target itself does not exist, this function creates an alias record for the target and returns `fnfErr`. Any other return value indicates that this function did not create an alias record.

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

Declared In
`Aliases.h`

FSResolveAlias

Returns an `FSRef` to the single most likely target of an alias record.

```
OSErr FSResolveAlias (
   const FSRef *fromFile,
   AliasHandle alias,
   FSRef *target,
   Boolean *wasChanged
);
```

Parameters

`fromFile`

A pointer to the starting point for a relative search. If you pass `NULL` in this parameter, `FSResolveAlias` performs only an absolute search. If you pass a pointer to a valid `FSRef` in the `fromFile` parameter, `FSResolveAlias` performs a relative search for the target, followed by an absolute search only if the relative search fails. If you want to perform an absolute search followed by a relative search, you should use the function `FSMatchAliasBulk` (page 185).

`alias`

A handle to the alias record to be resolved and, if necessary, updated.

`target`

A pointer to an `FSRef`. On successful return, this `FSRef` describes the target of the alias record. This parameter must point to a valid `FSRef` structure.

`wasChanged`

A pointer to a Boolean value indicating, on return, whether the alias record in the *`alias`* parameter was updated because it contained some outdated information about the target. If it updates the alias record, `FSResolveAlias` sets the `wasChanged` parameter to `true`. Otherwise, it sets it to `false`. (`FSResolveAlias` never updates a minimal alias, so it never sets `wasChanged` to `true` when resolving a minimal alias.

Return Value
A result code. When it finds the specified volume and parent directory but fails to find the target file or directory in that location, `FSResolveAlias` returns `fnfErr`. Note that the `FSRef` in the `alias` parameter is not valid in this case.

Discussion
The `FSResolveAlias` function performs a fast search for the target of the alias. If the resolution is successful, `FSResolveAlias` returns (in the *`target`* parameter) the `FSRef` for the target file system object, updates the alias record if necessary, and reports (through the *`wasChanged`* parameter) whether the record was updated. If the target is on an unmounted AppleShare volume, `FSResolveAlias` automatically mounts the volume. If the target is on an unmounted ejectable volume, `FSResolveAlias` asks the user to insert the volume. The `FSResolveAlias` function exits after it finds one acceptable target.

After it identifies a target, FSResolveAlias compares some key information about the target with the information in the alias record. If the information differs, FSResolveAlias updates the record to match the target.

The FSResolveAlias function displays the standard dialogs when it needs input from the user, such as a name and password for mounting a remote volume. The user can cancel the resolution through these dialogs.

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

Related Sample Code
QTCarbonShell

Declared In
Aliases.h

FSResolveAliasFile

Resolves an alias contained in an alias file.

```
OSErr FSResolveAliasFile (
   FSRef *theRef,
   Boolean resolveAliasChains,
   Boolean *targetIsFolder,
   Boolean *wasAliased
);
```

Parameters

theRef

A pointer to the alias file you plan to open. If the function completes successfully, this FSRef describes to the file or the directory referred to by the alias file.

resolveAliasChains

A Boolean value. Set this parameter to TRUE if you want FSResolveAliasFile to resolve all aliases in a chain (for example, an alias file that refers to an alias file and so on), stopping only when it reaches the target file. Set this parameter to FALSE if you want to resolve only one alias file, even if the target is another alias file.

targetIsFolder

A pointer to a Boolean value. The FSResolveAliasFile function returns TRUE in this parameter if the FSRef in the parameter theRef points to a directory or a volume; otherwise, FSResolveAliasFile returns FALSE in this parameter.

wasAliased

A pointer to a Boolean value. The FSResolveAliasFile function returns TRUE in this parameter if the FSRef in the parameter theRef points to an alias; otherwise, FSResolveAliasFile returns FALSE in this parameter.

Return Value
A result code. When it finds the specified volume and parent directory but fails to find the target file or directory in that location, FSResolveAliasFile returns fnfErr.

Discussion
If your application bypasses the Finder when manipulating documents, it should check for and resolve aliases itself by using the FSResolveAliasFile function.

The FSResolveAliasFile function first checks the catalog file for the file or directory specified in the parameter *theRef* to determine whether it is an alias and whether it is a file or a directory. If the object is not an alias, FSResolveAliasFile leaves *theRef* unchanged, sets the *targetIsFolder* parameter to TRUE for a directory or volume and FALSE for a file, sets *wasAliased* to FALSE, and returns noErr. If the object is an alias, FSResolveAliasFile resolves it, places the target in the parameter *theRef*, and sets the *wasAliased* flag to TRUE.

If FSResolveAliasFile receives an error code while resolving an alias, it leaves the input parameters as they are and exits, returning an error code. FSResolveAliasFile can return any Resource Manager or File Manager errors.

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

Declared In
Aliases.h

FSResolveAliasFileWithMountFlags

Resolves an alias contained in an alias file.

```
OSErr FSResolveAliasFileWithMountFlags (
   FSRef *theRef,
   Boolean resolveAliasChains,
   Boolean *targetIsFolder,
   Boolean *wasAliased,
   unsigned long mountFlags
);
```

Parameters

theRef

A pointer to the alias file you plan to open. If the function completes successfully, this FSRef describes the file or the directory referred to by the alias file.

resolveAliasChains

A Boolean value. Set this parameter to TRUE if you want FSResolveAliasFileWithMountFlags to resolve all aliases in a chain (for example, an alias file that refers to an alias file and so on), stopping only when it reaches the target file. Set this parameter to FALSE if you want to resolve only one alias file, even if the target is another alias file.

targetIsFolder

A pointer to a Boolean value. The FSResolveAliasFileWithMountFlags function returns TRUE in this parameter if the FSRef in the parameter theRef points to a directory or a volume; otherwise, FSResolveAliasFileWithMountFlags returns FALSE in this parameter.

wasAliased

A pointer to a Boolean value. The FSResolveAliasFileWithMountFlags function returns TRUE in this parameter if the FSRef in the parameter theRef points to an alias; otherwise, FSResolveAliasFileWithMountFlags returns FALSE in this parameter.

mountFlags

Options controlling how the alias file is resolved. See "Volume Mount Options" (page 220) for a description of the values you can use here. Set this parameter to kResolveAliasFileNoUI to prevent any user interaction, including disk switch alerts, while the alias is being resolved.

Return Value
A result code.

Discussion
The function `FSResolveAliasFileWithMountFlags` is identical to `FSResolveAliasFile` (page 193) with the exception that it provides the *`mountFlags`* parameter, allowing callers additional control over how the alias file is resolved.

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

Declared In
`Aliases.h`

FSResolveAliasWithMountFlags

Returns an `FSRef` to the target of an alias.

```
OSErr FSResolveAliasWithMountFlags (
   const FSRef *fromFile,
   AliasHandle inAlias,
   FSRef *target,
   Boolean *wasChanged,
   unsigned long mountFlags
);
```

Parameters

`fromFile`
A pointer to the starting point for a relative search. If you pass `NULL` in this parameter, `FSResolveAliasWithMountFlags` performs an absolute search. If you pass a pointer to a valid `FSRef` in the `fromFile` parameter, `FSResolveAliasWithMountFlags` performs a relative search for the target, followed by an absolute search only if the relative search fails. If you want to perform an absolute search followed by a relative search, you should use the function `FSMatchAliasBulk` (page 185).

`inAlias`
A handle to the alias record to be resolved and, if necessary, updated.

`target`
A pointer to an `FSRef` structure. On successful return, this `FSRef` refers to the target of the alias record. This parameter must point to a valid `FSRef` structure.

`wasChanged`
A pointer to a Boolean value indicating, on return, whether the alias record to be resolved was updated because it contained some outdated information about the target. If it updates the alias record, `FSResolveAliasWithMountFlags` sets the `wasChanged` parameter to `true`. Otherwise, it sets it to `false`. (`FSResolveAliasWithMountFlags` never updates a minimal alias, so it never sets `wasChanged` to `true` when resolving a minimal alias.

`mountFlags`
Options controlling how the alias is resolved. See "Volume Mount Options" (page 220) for a description of the values you can use here. Set this parameter to `kResolveAliasFileNoUI` to prevent any user interaction while the alias is being resolved.

Return Value
A result code.

Discussion
The function `FSResolveAliasWithMountFlags` is identical to `FSResolveAlias` (page 192) with the exception that it provides the `mountFlags` parameter, allowing callers additional control over how the alias is resolved.

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

Declared In
`Aliases.h`

FSUpdateAlias

Updates an alias record for a specified target.

```
OSErr FSUpdateAlias (
   const FSRef *fromFile,
   const FSRef *target,
   AliasHandle alias,
   Boolean *wasChanged
);
```

Parameters

`fromFile`
A pointer to the starting point for a relative search. You may pass `NULL` if you do not need relative search information in the alias record. The two files or directories specified in the `fromFile` and `target` parameters must reside on the same volume.

`target`
A pointer to the target of the alias record.

`alias`
A handle to the alias record to be updated.

`wasChanged`
A pointer to a Boolean value that, on output, indicates whether the newly constructed alias record is different from the old one. If the new record is exactly the same as the old one, the value is `false`. Otherwise, the value is `true`. Check this parameter to determine whether you need to save an updated record.

Return Value
A result code.

Discussion
This function rebuilds the entire alias record and fills it in as the `FSNewAlias` function would. The `FSUpdateAlias` function always creates a complete alias record. When you use `FSUpdateAlias` to update a minimal alias record, you convert the minimal record to a complete record.

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

Declared In
`Aliases.h`

GetAliasInfo

Gets information from an alias record without actually resolving the record. (Deprecated in Mac OS X v10.3. Use `FSCopyAliasInfo` (page 181) instead.)

```
OSErr GetAliasInfo (
   AliasHandle alias,
   AliasInfoType index,
   Str63 theString
);
```

Parameters

alias

A handle to the alias record to be read.

index

The kind of information to be retrieved. If the value of `index` is a positive integer, `GetAliasInfo` retrieves the parent directory that has the same hierarchical level above the target as the index parameter (for example, an `index` value of 2 returns the name of the parent directory of the target's parent directory). You can therefore assemble the names of the target and all of its parent directories by making repeated calls to `GetAliasInfo` with incrementing index values, starting with a value of 0. When the value of `index` is greater than the number of levels between the target and the root, `GetAliasInfo` returns an empty string. You can also set the `index` parameter to one of the values described in "Information Type Constants" (page 222).

theString

A string that, on return, holds the requested information.

Return Value

A result code.

Discussion

The `GetAliasInfo` function returns the information stored in the alias record, which might not be current. To ensure that the information is current, you can resolve and update the alias record before calling `GetAliasInfo`.

The `GetAliasInfo` function cannot provide all kinds of information about a minimal alias.

Special Considerations

Use the `FSCopyAliasInfo` (page 181) function instead of `GetAliasInfo`. `GetAliasInfo` does not reliably return information for aliases to items on POSIX file systems. In addition, `GetAliasInfo` does not support unicode names or names longer than 32 bytes. If the name of the alias target is longer than 32 bytes, the name is truncated and the file ID and extension (if any) are appended before the name is returned by `GetAliasInfo`.

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

`Aliases.h`

GetAliasSize

Gets the size of an alias record referenced by a handle.

```
Size GetAliasSize (
   AliasHandle alias
);
```

Parameters

alias

A handle to the alias record from which to get the information.

Return Value

The size of the alias record.

Discussion

The returned size is smaller than the size returned by the function `GetHandleSize` if any custom data is added. This routine is thread safe.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`Aliases.h`

GetAliasSizeFromPtr

Gets the size of an alias record referenced by a pointer.

```
Size GetAliasSizeFromPtr (
   const AliasRecord *alias
);
```

Parameters

alias

A pointer to the alias record from which to get the information.

Return Value

The size of the alias record.

Discussion

This routine is thread safe.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`Aliases.h`

GetAliasUserType

Gets the user type for an alias record referenced by a handle.

```
OSType GetAliasUserType (
   AliasHandle alias
);
```

Parameters

alias

A handle to the alias record from which to get the user type.

Return Value
The user type associated with the alias.

Discussion
This routine is thread safe.

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

Declared In
`Aliases.h`

GetAliasUserTypeFromPtr

Gets the user type for the alias record referenced by a pointer.

```
OSType GetAliasUserTypeFromPtr (
   const AliasRecord *alias
);
```

Parameters

alias

A pointer to the alias record from which to get the user type.

Return Value
The user type associated with the alias.

Discussion
This routine is thread safe.

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

Declared In
`Aliases.h`

InvokeAliasFilterUPP

Calls your alias filtering callback function.

```
Boolean InvokeAliasFilterUPP (
   CInfoPBPtr cpbPtr,
   Boolean *quitFlag,
   Ptr myDataPtr,
   AliasFilterUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeAliasFilterUPP`, as the system calls your alias filtering callback for you.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Aliases.h`

IsAliasFile

Determines whether a file system object is an alias file, a data file, or a folder. (Deprecated in Mac OS X v10.4. Use `FSIsAliasFile` (page 183) instead.)

```
OSErr IsAliasFile (
   const FSSpec *fileFSSpec,
   Boolean *aliasFileFlag,
   Boolean *folderFlag
);
```

Parameters

fileFSSpec
A pointer to a file specification structure describing a file.

aliasFileFlag
A pointer to a Boolean variable. On return, a value of `TRUE` indicates that the object specified in the *fileRef* parameter is an alias file. A value of `FALSE` indicates that the object is not an alias file.

folderFlag
A pointer to a Boolean variable. On return, a value of `TRUE` indicates that the object specified in the *fileRef* parameter is a folder. A value of `FALSE` indicates that the object is a file.

Return Value
A result code.

Discussion
This function determines whether a file is an alias file.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Aliases.h`

MatchAlias

Identifies a list of possible matches for an alias and passes the list through an optional selection filter. The filter can return more than one possible match. (Deprecated in Mac OS X v10.4. Use `FSMatchAliasBulk` (page 185) instead.)

```
OSErr MatchAlias (
   const FSSpec *fromFile,
   unsigned long rulesMask,
   AliasHandle alias,
   short *aliasCount,
   FSSpecArrayPtr aliasList,
   Boolean *needsUpdate,
   AliasFilterUPP aliasFilter,
   void *yourDataPtr
);
```

Parameters

fromFile

A pointer to the starting point for a relative search. If you do not want `MatchAlias` to perform a relative search, set `fromFile` to `NULL`. If you want `MatchAlias` to perform a relative search, pass a pointer to a file system specification structure that describes the starting point for the search.

rulesMask

A set of rules to guide the resolution. Pass the sum of all of the rules you want to invoke. For a description of the values you can use in this parameter, see "Matching Constants" (page 220).

alias

A handle to the alias record to be resolved.

aliasCount

On input, a pointer to the maximum number of possible matches to return. On output, the actual number of matches returned.

aliasList

A pointer to the array that holds the results of the search, a list of possible candidates.

needsUpdate

A pointer to a Boolean flag that indicates whether the alias record to be resolved needs to be updated.

aliasFilter

An application-defined filter function. The Alias Manager executes this function each time it identifies a possible match and after the search has continued for three seconds without a match. Your filter function returns a Boolean value that determines whether the possible match is discarded (`true`) or added to the list of possible targets (`false`). It can also terminate the search by setting the variable parameter `quitFlag`. See `AliasFilterProcPtr` (page 215) for a description of the filter function.

yourDataPtr

A pointer to data to be passed to the filter function. The `yourDataPtr` parameter can point to any data your application might need in the filter function.

Return Value

A result code.

Discussion

If `MatchAlias` finds the parent directory on the correct volume but does not find the target, it sets the *aliasCount* parameter to 1, puts the file system specification structure for the target in the results list, and returns `fnfErr`. The `FSSpec` structure is valid, although the object it describes does not exist. This information

is intended as a "hint" that lets you explore possible solutions to the resolution failure. You can, for example, use the `FSSpec` structure and the File Manager function `FSpCreate` to create a replacement for a missing file.

After it identifies a target, `MatchAlias` compares some key information about the target with the same information in the record. If the information does not match, `MatchAlias` sets the *`needsUpdate`* flag to `true`. The key information is

- the name of the target
- the directory ID of the target's parent
- the file ID or directory ID of the target
- the name and creation date of the volume on which the target resides

The `MatchAlias` function also sets the *`needsUpdate`* flag to `true` if it identifies a list of possible matches rather than a single match or if `kARMsearchRelFirst` is set in the `rulesMask` parameter but the target is identified through either an absolute search or an exhaustive search. Otherwise, the `MatchAlias` function sets the *`needsUpdate`* flag to `false`. `MatchAlias` always sets the *`needsUpdate`* flag to `false` when resolving an alias created by `NewAliasMinimal`. If you want to update the alias record to reflect the final results of the resolution, call `UpdateAlias`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Aliases.h`

MatchAliasNoUI

Identifies a list of possible matches for an alias without any user interaction. (Deprecated in Mac OS X v10.5. Use `FSMatchAliasBulk` (page 185) with the `kARMNoUI` flag instead.)

```
OSErr MatchAliasNoUI (
   const FSSpec *fromFile,
   unsigned long rulesMask,
   AliasHandle alias,
   short *aliasCount,
   FSSpecArrayPtr aliasList,
   Boolean *needsUpdate,
   AliasFilterUPP aliasFilter,
   void *yourDataPtr
);
```

Parameters

`fromFile`

A pointer to the starting point for a relative search. If you do not want `MatchAliasNoUI` to perform a relative search, set `fromFile` to `NULL`. If you want `MatchAliasNoUI` to perform a relative search, pass a pointer to a file system specification structure that describes the starting point for the search.

rulesMask
: A set of rules to guide the resolution. Pass the sum of all of the rules you want to invoke. For a description of the values you can use in this parameter, see "Matching Constants" (page 220).

alias
: A handle to the alias record to be resolved.

aliasCount
: On input, a pointer to the maximum number of possible matches to return. On output, the actual number of matches returned.

aliasList
: A pointer to the array of `FSSpec` structures that holds, on return, the results of the search, a list of possible candidates.

needsUpdate
: A pointer to a Boolean flag that, on return, indicates whether the alias record needs to be updated.

aliasFilter
: An application-defined filter function. The Alias Manager executes this function each time it identifies a possible match and after the search has continued for three seconds without a match. Your filter function returns a Boolean value that determines whether the possible match is discarded (`true`) or added to the list of possible targets (`false`). It can also terminate the search by setting the variable parameter `quitFlag`. See `AliasFilterProcPtr` (page 215) for a description of the filter function.

yourDataPtr
: A pointer to data to be passed to the filter function. The `yourDataPtr` parameter can point to any data your application might need in the filter function.

Return Value
A result code.

Discussion
The `MatchAliasNoUI` function operates in the same way as the `MatchAlias` function; however, it does not present an interface to the user. See the discussion of `MatchAlias` (page 201) for more information.

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
`Aliases.h`

NewAlias

Creates a complete alias record. (Deprecated in Mac OS X v10.4. Use `FSNewAlias` (page 188) instead.)

```
OSErr NewAlias (
   const FSSpec *fromFile,
   const FSSpec *target,
   AliasHandle *alias
);
```

Parameters

fromFile

A pointer to the starting point for a relative search. If you do not need relative search information in the alias record, pass a `fromFile` value of `NULL`. If you want `NewAlias` to record relative search information, pass a pointer to a valid `FSSpec` structure in this parameter. The files or directories specified in the `fromFile` and `target` parameters must reside on the same volume.

target

A pointer to an `FSSpec` structure for the target of the alias record.

alias

A pointer to an alias handle. On return, this handle refers to the newly created alias record. If the function fails to create an alias record, it sets *alias* to `NULL`.

Return Value
A result code.

Discussion
The `NewAlias` function creates an alias record that describes the specified target. It allocates the storage, fills in the record, and puts a record handle to that storage in the *alias* parameter. `NewAlias` always records the name and file or directory ID of the target, its creation date, the parent directory name and ID, and the volume name and creation date. It also records the full pathname of the target and a collection of other information relevant to locating the target, verifying the target, and mounting the target's volume, if necessary. You can have `NewAlias` store relative search information as well by supplying a starting point for a relative search.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Aliases.h`

NewAliasFilterUPP

Creates a new universal procedure pointer (UPP) to an alias filtering callback function.

```
AliasFilterUPP NewAliasFilterUPP (
   AliasFilterProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your alias filtering callback function. For more information, see `AliasFilterProcPtr` (page 215).

Return Value
On return, a UPP to the alias filtering callback function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Aliases.h`

NewAliasMinimal

Creates a short alias record quickly. (Deprecated in Mac OS X v10.4. Use `FSNewAliasMinimal` (page 190) instead.)

```
OSErr NewAliasMinimal (
   const FSSpec *target,
   AliasHandle *alias
);
```

Parameters

target
A pointer to the target of the alias record.

alias
A pointer to an alias handle. On return, this handle refers to the newly created alias record. If the function fails to create an alias record, it sets `alias` to `NULL`.

Return Value
A result code.

Discussion
The `NewAliasMinimal` function creates an alias record that contains only the minimum information necessary to describe the target: the target name, the parent directory ID, the volume name and creation date, and the volume mounting information. The `NewAliasMinimal` function uses the standard alias record data structure, but it fills in only parts of the record.

The `ResolveAlias` (page 206) function never updates a minimal alias record.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Aliases.h`

NewAliasMinimalFromFullPath

Creates an alias record that contains only the full pathname of the target. (Deprecated in Mac OS X v10.4. Use `FSNewAliasMinimal` (page 190) or `FSNewAliasMinimalUnicode` (page 190) instead.)

```
OSErr NewAliasMinimalFromFullPath (
   short fullPathLength,
   const void *fullPath,
   ConstStr32Param zoneName,
   ConstStr31Param serverName,
   AliasHandle *alias
);
```

Parameters

fullPathLength

The number of characters in the full pathname of the target.

fullPath

A pointer to a buffer that contains the full pathname of the target. The full pathname starts with the name of the volume, includes all of the directory names in the path to the target, and ends with the target name. (For a description of pathnames, see the documentation for the File Manager.)

zoneName

The AppleTalk zone name of the AppleShare volume on which the target resides. Set this parameter to a null string if you do not need it.

serverName

The AppleTalk server name of the AppleShare volume on which the target resides. Set this parameter to a null string if you do not need it.

alias

A pointer to an alias handle. On return, this handle refers to the newly created alias record. If the function fails to create an alias record, it sets `alias` to `NULL`.

Return Value
A result code.

Discussion
The `NewAliasMinimalFromFullPath` function creates an alias record that identifies the target by full pathname. You can call `NewAliasMinimalFromFullPath` to create an alias record for a file that doesn't exist or that resides on an unmounted volume.

The `NewAliasMinimalFromFullPath` function uses the standard alias record data structure, but it fills in only the information provided in the input parameters. You can therefore use `NewAliasMinimalFromFullPath` to create alias records for targets on unmounted volumes.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Aliases.h`

ResolveAlias

Identifies the single most likely target of an alias record. (Deprecated in Mac OS X v10.4. Use `FSResolveAlias` (page 192) instead.)

```
OSErr ResolveAlias (
   const FSSpec *fromFile,
   AliasHandle alias,
   FSSpec *target,
   Boolean *wasChanged
);
```

Parameters

fromFile

A pointer to the starting point for a relative search. If you pass a `fromFile` parameter of `NULL`, `ResolveAlias` performs only an absolute search. If you pass a pointer to a valid `FSSpec` structure in the `fromFile` parameter, `ResolveAlias` performs a relative search for the target, followed by an absolute search only if the relative search fails. If you want to perform an absolute search followed by a relative search, you must use the `MatchAlias` function.

alias

A handle to the alias record to be resolved and, if necessary, updated.

target

A pointer to the target of the alias record. This parameter must be a valid `FSSpec` structure.

wasChanged

A pointer to a Boolean value indicating whether the alias record to be resolved was updated because it contained some outdated information about the target. If it updates the alias record, `ResolveAlias` sets the `wasChanged` parameter to `true`. Otherwise, it sets it to `false`. (`ResolveAlias` never updates a minimal alias, so it never sets `wasChanged` to `true` when resolving a minimal alias.

Return Value
A result code.

Discussion
The `ResolveAlias` function performs a fast search for the target of the alias. If the resolution is successful, `ResolveAlias` returns (in the *`target`* parameter) the `FSSpec` structure for the target file system object, updates the alias record if necessary, and reports (through the *`wasChanged`* parameter) whether the record was updated. If the target is on an unmounted AppleShare volume, `ResolveAlias` automatically mounts the volume. If the target is on an unmounted ejectable volume, `ResolveAlias` asks the user to insert the volume. The `ResolveAlias` function exits after it finds one acceptable target.

After it identifies a target, `ResolveAlias` compares some key information about the target with the information in the alias record. (The description of the `MatchAlias` (page 201) function lists the key information.) If the information differs, `ResolveAlias` updates the record to match the target.

When it finds the specified volume and parent directory but fails to find the target file or directory in that location, `ResolveAlias` returns a result code of `fnfErr` and fills in the *`target`* parameter with a complete `FSSpec` structure describing the target (that is, the volume reference number, parent directory ID, and filename or folder name). The `FSSpec` structure is valid, although the object it describes does not exist. This information is intended as a "hint" that lets you explore possible solutions to the resolution failure. You can, for example, pass the `FSSpec` structure to the File Manager function `FSpCreate` to create a replacement for a missing file.

The `ResolveAlias` function displays the standard dialog boxes when it needs input from the user, such as a name and password for mounting a remote volume. The user can cancel the resolution through these dialog boxes.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Aliases.h`

ResolveAliasFile

Resolves an alias contained in an alias file. (Deprecated in Mac OS X v10.4. Use `FSResolveAliasFile` (page 193) instead.)

```
OSErr ResolveAliasFile (
   FSSpec *theSpec,
   Boolean resolveAliasChains,
   Boolean *targetIsFolder,
   Boolean *wasAliased
);
```

Parameters

theSpec

A pointer to the alias file you plan to open. If the function completes successfully, this `FSSpec` refers to the file or the directory that was referred to by the alias file.

resolveAliasChains

A Boolean value. Set this parameter to `TRUE` if you want `ResolveAliasFile` to resolve all aliases in a chain (for example, an alias file that refers to an alias file and so on), stopping only when it reaches the target file. Set this parameter to `FALSE` if you want to resolve only one alias file, even if the target is another alias file.

targetIsFolder

A return parameter only. The `ResolveAliasFile` function returns `TRUE` in this parameter if the file specification structure in the parameter `theSpec` points to a directory or a volume; otherwise, `ResolveAliasFile` returns `FALSE` in this parameter.

wasAliased

A return parameter only. The `ResolveAliasFile` function returns `TRUE` in this parameter if the file specification structure in the parameter `theSpec` points to an alias; otherwise, `ResolveAliasFile` returns `FALSE` in this parameter.

Return Value
A result code.

Discussion
If your application bypasses the Finder when manipulating documents, it should check for and resolve aliases itself by using the `ResolveAliasFile` function.

The `ResolveAliasFile` function first checks the catalog file for the file or directory specified in the parameter *theSpec* to determine whether it is an alias and whether it is a file or a directory. If the object is not an alias, `ResolveAliasFile` leaves *theSpec* unchanged, sets the *targetIsFolder* parameter to `TRUE` for a directory or volume and `FALSE` for a file, sets *wasAliased* to `FALSE`, and returns `noErr`. If the object is an alias, `ResolveAliasFile` resolves it, places the target in the parameter *theSpec*, and sets the *wasAliased* flag to `TRUE`.

When `ResolveAliasFile` finds the specified volume and parent directory but fails to find the target file or directory in that location, `ResolveAliasFile` returns a result code of `fnfErr` and fills in the parameter *theSpec* with a complete file system specification structure describing the target (that is, its volume reference number, parent directory ID, and filename or folder name). The file system specification structure is valid,

although the object it describes does not exist. This information is intended as a "hint" that lets you explore possible solutions to the resolution failure. You can, for example, use the file system specification structure to create a replacement for a missing file with the File Manager function `FSpCreate`.

If `ResolveAliasFile` receives an error code while resolving an alias, it leaves the input parameters as they are and exits, returning an error code. `ResolveAliasFile` can return any Resource Manager or File Manager errors.

Special Considerations

Before calling the `ResolveAliasFile` function, you should make sure that it is available by using the `Gestalt` function with the `gestaltAliasMgrAttr` selector.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Aliases.h`

ResolveAliasFileWithMountFlags

Resolves an alias contained in an alias file. (Deprecated in Mac OS X v10.5. Use `FSResolveAliasFileWithMountFlags` (page 194) instead.)

```
OSErr ResolveAliasFileWithMountFlags (
   FSSpec *theSpec,
   Boolean resolveAliasChains,
   Boolean *targetIsFolder,
   Boolean *wasAliased,
   unsigned long mountFlags
);
```

Parameters

`theSpec`

A pointer to the alias file you plan to open. If the function completes successfully, this `FSSpec` refers to the file or the directory that was referred to by the alias file.

`resolveAliasChains`

A Boolean value. Set this parameter to `TRUE` if you want `ResolveAliasFileWithMountFlags` to resolve all aliases in a chain (for example, an alias file that refers to an alias file and so on), stopping only when it reaches the target file. Set this parameter to `FALSE` if you want to resolve only one alias file, even if the target is another alias file.

`targetIsFolder`

A return parameter only. The `ResolveAliasFileWithMountFlags` function returns `TRUE` in this parameter if the file specification structure in the parameter `theSpec` points to a directory or a volume; otherwise, `ResolveAliasFileWithMountFlags` returns `FALSE` in this parameter.

`wasAliased`

A return parameter only. The `ResolveAliasFileWithMountFlags` function returns `TRUE` in this parameter if the file specification structure in the parameter `theSpec` points to an alias; otherwise, `ResolveAliasFileWithMountFlags` returns `FALSE` in this parameter.

mountFlags

Options controlling how the alias file is resolved. See "Volume Mount Options" (page 220) for a description of the values you can use here. Set this parameter to `kResolveAliasFileNoUI` to prevent any user interaction while the alias is being resolved.

Return Value
A result code.

Discussion
The function `ResolveAliasFileWithMountFlags` is identical to `ResolveAliasFile` (page 208) with the exception that it provides the *mountFlags* parameter.

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
`Aliases.h`

ResolveAliasFileWithMountFlagsNoUI

Resolves an alias file without any user interaction. (Deprecated in Mac OS X v10.4. Use `FSResolveAliasFileWithMountFlags` (page 194) with the `kResolveAliasFileNoUI` flag instead.)

```
OSErr ResolveAliasFileWithMountFlagsNoUI (
   FSSpec *theSpec,
   Boolean resolveAliasChains,
   Boolean *targetIsFolder,
   Boolean *wasAliased,
   unsigned long mountFlags
);
```

Parameters

theSpec

A pointer to the alias file you plan to open. If the function completes successfully, this `FSSpec` refers to the file or the directory that was referred to by the alias file.

resolveAliasChains

A Boolean value. Set this parameter to `TRUE` if you want `ResolveAliasFileWithMountFlagsNoUI` to resolve all aliases in a chain (for example, an alias file that refers to an alias file and so on), stopping only when it reaches the target file. Set this parameter to `FALSE` if you want to resolve only one alias file, even if the target is another alias file.

targetIsFolder

A return parameter only. The `ResolveAliasFileWithMountFlagsNoUI` function returns `TRUE` in this parameter if the file specification structure in the parameter `theSpec` points to a directory or a volume; otherwise, `ResolveAliasFileWithMountFlagsNoUI` returns `FALSE` in this parameter.

wasAliased

A return parameter only. The `ResolveAliasFileWithMountFlagsNoUI` function returns `TRUE` in this parameter if the file specification structure in the parameter `theSpec` points to an alias; otherwise, `ResolveAliasFileWithMountFlagsNoUI` returns `FALSE` in this parameter.

mountFlags

Options controlling how the alias file is resolved. See "Volume Mount Options" (page 220) for a description of the values you can use here. Set this parameter to `kResolveAliasFileNoUI` to prevent any user interaction, including disk switch alerts, while the alias is being resolved.

Return Value
A result code.

Discussion
The function `ResolveAliasFileWithMountFlagsNoUI` is identical to `ResolveAliasFile` (page 208) with the exception that it presents no interface to the user.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Aliases.h`

ResolveAliasWithMountFlags

Identifies the target of an alias. (Deprecated in Mac OS X v10.4. Use `FSResolveAliasWithMountFlags` (page 195) instead.)

```
OSErr ResolveAliasWithMountFlags (
   const FSSpec *fromFile,
   AliasHandle alias,
   FSSpec *target,
   Boolean *wasChanged,
   unsigned long mountFlags
);
```

Parameters

fromFile

A pointer to the starting point for a relative search. If you pass `NULL` in this parameter, `ResolveAliasWithMountFlags` performs only an absolute search. If you pass a pointer to a valid `FSSpec` structure in the `fromFile` parameter, `ResolveAliasWithMountFlags` performs a relative search for the target, followed by an absolute search only if the relative search fails. If you want to perform an absolute search followed by a relative search, you must use the `MatchAlias` function.

alias

A handle to the alias record to be resolved and, if necessary, updated.

target

A pointer to an `FSSpec` structure. On return, this `FSSpec` identifies the target of the alias record. This parameter must point to a valid `FSSpec` structure.

wasChanged

A pointer to a Boolean value indicating, on return, whether the alias record to be resolved was updated because it contained some outdated information about the target. If it updates the alias record, `ResolveAliasWithMountFlags` sets the `wasChanged` parameter to `true`. Otherwise, it sets it to `false`. (`ResolveAliasWithMountFlags` never updates a minimal alias, so it never sets `wasChanged` to `true` when resolving a minimal alias.

mountFlags
Options controlling how the alias is resolved. See "Volume Mount Options" (page 220) for a description of the values you can use here. Set this parameter to `kResolveAliasFileNoUI` to prevent any user interaction while the alias is being resolved.

Return Value
A result code.

Discussion
The function `ResolveAliasWithMountFlags` is identical to `ResolveAlias` (page 206) with the exception that it provides the `mountFlags` parameter, allowing callers additional control over how the alias is resolved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Aliases.h`

SetAliasUserType

Sets the user type for an alias record referenced by a handle.

```
void SetAliasUserType (
   AliasHandle alias,
   OSType userType
);
```

Parameters
alias
A handle to the alias record for which to set the user type.
userType
The user type associated with the alias.

Discussion
This routine is thread safe.

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

Declared In
`Aliases.h`

SetAliasUserTypeWithPtr

Sets the user type for the alias record referenced by a pointer.

```
void SetAliasUserTypeWithPtr (
   AliasPtr alias,
   OSType userType
);
```

Parameters

alias

A pointer to the alias record for which to set the user type.

userType

The user type associated with the alias.

Discussion
This routine is thread safe.

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

Declared In
`Aliases.h`

UpdateAlias

Updates an alias record. (Deprecated in Mac OS X v10.4. Use `FSUpdateAlias` (page 196) instead.)

```
OSErr UpdateAlias (
   const FSSpec *fromFile,
   const FSSpec *target,
   AliasHandle alias,
   Boolean *wasChanged
);
```

Parameters

fromFile

A pointer to the starting point for a relative search. If you do not need relative search information in the record, pass a `fromFile` value of `NULL`. If you want `UpdateAlias` to record relative search information, pass a pointer to a valid `FSSpec` structure in this parameter.

target

A pointer to the target of the alias record.

alias

A handle to the alias record to be updated.

wasChanged

A pointer to a Boolean value indicating whether the newly constructed alias record is exactly the same as the old one. If the new record is the same as the old one, `UpdateAlias` sets the `wasChanged` parameter to `false`. Otherwise, it sets it to `true`. Check this parameter to determine whether you need to save an updated record.

Return Value
A result code.

Discussion
The `UpdateAlias` function rebuilds the entire alias record and fills it in as the `NewAlias` function would.

The `UpdateAlias` function always creates a complete alias record. When you use `UpdateAlias` to update a minimal alias record, you convert the minimal record to a complete record.

Special Considerations

The two files or directories, specified in the `fromFile` and `target` parameters, must reside on the same volume.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Aliases.h`

Callbacks

AliasFilterProcPtr

Defines a pointer to an alias filtering callback function that filters out possible targets identified by the `FSMatchAlias` (page 184) function.

```
typedef Boolean (*AliasFilterProcPtr) (
    CInfoPBPtr cpbPtr,
    Boolean * quitFlag,
    Ptr myDataPtr
);
```

If you name your function `MyAliasFilterCallback`, you would declare it like this:

```
Boolean MyAliasFilterCallback (
    CInfoPBPtr cpbPtr,
    Boolean * quitFlag,
    Ptr myDataPtr
);
```

Parameters

`cpbPtr`

A pointer to a catalog information parameter block. When your function is called, the `cpbPtr` parameter points to the catalog information parameter block of the possible match (returned by the File Manager function `PBGetCatInfo`).

`quitFlag`

On exit, set this to `true` if you want to terminate the search.

`myDataPtr`

A pointer to any customized data that your application passed when it called `FSMatchAlias` (page 184). This parameter allows your filter function to access any data that your application has set up on its own.

Return Value
Your function should return `true` to indicate that the possible match is to be discarded, or `false` to indicate that the possible match is to be added to the list of possible targets.

Discussion
You can write your own filter function to examine possible targets identified by the `FSMatchAlias` function. The `FSMatchAlias` function calls your filter function each time it identifies a possible match.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Aliases.h`

FSAliasFilterProcPtr

Defines a pointer to an alias filtering callback function that filters out possible targets identified by the `FSMatchAliasBulk` (page 185) function.

```
typedef Boolean (*FSAliasFilterProcPtr) (
    FSRef *ref,
    Boolean *quitFlag,
    Ptr myDataPtr
);
```

If you name your function `MyFSAliasFilterCallback`, you would declare it like this:

```
Boolean MyAliasFilterCallback (
    FSRef *ref,
    Boolean *quitFlag,
    Ptr myDataPtr
);
```

Parameters

ref
A pointer to a file system object. When your function is called, the *ref* parameter points to the possible match.

quitFlag
On output, set this Boolean flag to `true` if you want to terminate the search.

myDataPtr
A pointer to any customized data that your application passed when it called `FSMatchAliasBulk` (page 185). This parameter allows your filter function to access any data that your application has set up on its own.

Return Value
Your function should return `true` to indicate that the possible match is to be discarded, or `false` to indicate that the possible match is to be added to the list of possible targets.

Discussion
You can write your own filter function to examine possible targets identified by the `FSMatchAliasBulk` function. The `FSMatchAliasBulk` function calls your filter function each time it identifies a possible match.

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

Declared In
`Aliases.h`

Data Types

AliasInfoType

Defines the alias record information type used in the index parameter of `GetAliasInfo`.

```
typedef short AliasInfoType;
```

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

Declared In
`Aliases.h`

AliasFilterUPP

Defines a universal procedure pointer (UPP) to an alias filtering function.

```
typedef AliasFilterProcPtr AliasFilterUPP;
```

Discussion
See `AliasFilterProcPtr` (page 215) for more information on alias filtering functions.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Aliases.h`

AliasRecord

Defines an alias record.

```
struct AliasRecord {
    OSType userType;
    unsigned short aliasSize;
};
typedef struct AliasRecord              AliasRecord;
typedef AliasRecord *                   AliasPtr;
typedef AliasPtr *                      AliasHandle;
```

Fields

`userType`

A 4-byte field that can contain application-specific data. When an alias record is created, this field contains 0. Your application can use this field for its own purposes.

`aliasSize`

The size, in bytes, assigned to the alias record at the time of its creation or updating. This is the total size of the record, including the `userType` and `aliasSize` fields, as well as the variable-length data that is private to the Alias Manager.

Discussion

The Alias Manager uses alias records to store information that allows it to locate an object in the file system.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Aliases.h`

FSAliasInfo

Defines an information block passed to the `FSCopyAliasInfo` function.

```
struct FSAliasInfo {
    UTCDateTime volumeCreateDate;
    UTCDateTime targetCreateDate;
    OSType fileType;
    OSType fileCreator;
    UInt32 parentDirID;
    UInt32 nodeID;
    UInt16 filesystemID;
    UInt16 signature;
    Boolean volumeIsBootVolume;
    Boolean volumeIsAutomounted;
    Boolean volumeIsEjectable;
    Boolean volumeHasPersistentFileIDs;
    Boolean isDirectory;
};
typedef struct FSAliasInfo FSAliasInfo;
typedef FSAliasInfo * FSAliasInfoPtr;
```

Fields

`volumeCreateDate`

The creation date of the volume on which the alias target resides.

`targetCreateDate`

The creation date of the alias target.

`fileType`
The file type of the target.

`fileCreator`
The creator code of the target.

`parentDirID`
The directory ID of the target's parent directory.

`nodeID`
The ID of the file or directory that is the alias target.

`filesystemID`
The filesystem ID.

`signature`
The volume signature of the volume on which the target resides.

`volumeIsBootVolume`
A Boolean value indicating whether the volume is the boot volume.

`volumeIsAutomounted`
A Boolean value indicating whether the volume is automounted.

`volumeIsEjectable`
A Boolean value indicating whether the volume is ejectable.

`volumeHasPersistentFileIDs`
A Boolean value indicating whether the volume has persistent file ID's.

`isDirectory`
A Boolean value indicating whether the alias target is a directory.

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

Declared In
`Aliases.h`

Constants

Alias Information Masks

Returned by the `FSCopyAliasInfo` function to indicate which fields of the alias information structure contain valid data.

```
typedef UInt32 FSAliasInfoBitmap;
enum {
    kFSAliasInfoNone = 0x00000000,
    kFSAliasInfoVolumeCreateDate = 0x00000001,
    kFSAliasInfoTargetCreateDate = 0x00000002,
    kFSAliasInfoFinderInfo = 0x00000004,
    kFSAliasInfoIsDirectory = 0x00000008,
    kFSAliasInfoIDs = 0x00000010,
    kFSAliasInfoFSInfo = 0x00000020,
    kFSAliasInfoVolumeFlags = 0x00000040
};
```

Constants

`kFSAliasInfoNone`
None of the alias information is valid.

Available in Mac OS X v10.2 and later.

Declared in `Aliases.h`.

`kFSAliasInfoVolumeCreateDate`
The volume creation date in the `volumeCreateDate` field is valid.

Available in Mac OS X v10.2 and later.

Declared in `Aliases.h`.

`kFSAliasInfoTargetCreateDate`
The creation date of the alias target, in the `targetCreateDate` field, is valid.

Available in Mac OS X v10.2 and later.

Declared in `Aliases.h`.

`kFSAliasInfoFinderInfo`
The file type and creator information, in the `fileType` and `fileCreator` fields, is valid.

Available in Mac OS X v10.2 and later.

Declared in `Aliases.h`.

`kFSAliasInfoIsDirectory`
The information in the `isDirectory` field is valid.

Available in Mac OS X v10.2 and later.

Declared in `Aliases.h`.

`kFSAliasInfoIDs`
The parent directory ID and alias target ID, in the `parentDirID` and `nodeID` fields, are valid.

Available in Mac OS X v10.2 and later.

Declared in `Aliases.h`.

`kFSAliasInfoFSInfo`
The filesystem ID and signature, in the `filesystemID` and `signature` fields, are valid.

Available in Mac OS X v10.2 and later.

Declared in `Aliases.h`.

`kFSAliasInfoVolumeFlags`
The volume information, in the `volumeIsBootVolume`, `volumeIsAutomounted`, `volumeIsEjectable`, and `volumeHasPersistentFileIDs` fields, is valid.

Available in Mac OS X v10.2 and later.

Declared in `Aliases.h`.

Volume Mount Options

Specify how an alias should be resolved.

```
enum {
    kResolveAliasFileNoUI = 0x00000001,
    kResolveAliasTryFileIDFirst = 0x00000002
};
```

Constants

`kResolveAliasFileNoUI`

The Alias Manager should resolve the alias without presenting a user interface.

Available in Mac OS X v10.0 and later.

Declared in `Aliases.h`.

`kResolveAliasTryFileIDFirst`

The Alias Manager should search for the alias target using file IDs before searching using the path.

Available in Mac OS X v10.2 and later.

Declared in `Aliases.h`.

Discussion

The `FSResolveAliasWithMountFlags` (page 195), `FSResolveAliasFileWithMountFlags` (page 194), `ResolveAliasWithMountFlags` (page 211), `ResolveAliasFileWithMountFlags` (page 209), and `ResolveAliasFileWithMountFlagsNoUI` (page 210) functions take these constants in the `mountFlags` parameter, allowing you to specify how the alias should be resolved.

Matching Constants

Specify the matching criteria for the alias matching functions.

```
enum {
    kARMMountVol = 0x00000001,
    kARMNoUI = 0x00000002,
    kARMMultVols = 0x00000008,
    kARMSearch = 0x00000100,
    kARMSearchMore = 0x00000200,
    kARMSearchRelFirst = 0x00000400,
    kARMTryFileIDFirst = 0x00000800
};
```

Constants

`kARMMountVol`

Automatically try to mount the target's volume if it is not mounted.

Available in Mac OS X v10.0 and later.

Declared in `Aliases.h`.

`kARMNoUI`

Stop if a search requires user interaction, such as a password dialog box when mounting a remote volume. If user interaction is needed and `kARMNoUI` is in effect, the search fails.

Available in Mac OS X v10.0 and later.

Declared in `Aliases.h`.

`kARMMultVols`
: Search all mounted volumes. The search begins with the volume on which the target resided when the record was created. When you specify a fast search of all mounted volumes, `MatchAlias` performs a formal fast search only on the volume described in the alias record. On all other volumes it looks for the target by ID or by name in the directory with the specified parent directory ID. When you specify an exhaustive search of multiple volumes, `MatchAlias` performs the same search on all volumes. When resolving an alias record created by `NewAliasMinimalFromFullPath`, `MatchAlias` ignores this flag.

  Available in Mac OS X v10.0 and later.

  Declared in `Aliases.h`.

`kARMSearch`
: Perform a fast search for the alias target. If `kARMSearchRelFirst` is not set, perform an absolute search first, followed by a relative search only if the value of the `fromFile` parameter is not `NULL` and the list of matches is not full.

  Available in Mac OS X v10.0 and later.

  Declared in `Aliases.h`.

`kARMSearchMore`
: Perform an exhaustive search for the alias target. On HFS volumes, the exhaustive search uses the File Manager function `PBCatSearch` to identify candidates with matching creation date, type, and creator. The `PBCatSearch` function is available only on HFS volumes and only on systems running version 7.0 or later. On MFS volumes or HFS volumes that do not support `PBCatSearch`, the exhaustive search makes a series of indexed calls to File Manager functions, using the same search criteria. If you set `kARMSearchMore` and either or both of `kARMSearch` and `kARMSearchRelFirst`, `MatchAlias` performs the fast search first.

  Available in Mac OS X v10.0 and later.

  Declared in `Aliases.h`.

`kARMSearchRelFirst`
: If `kARMSearch` is also set, perform a relative search before the absolute search. (If `kARMSearch` is also set and the target is found through the absolute search, `MatchAlias` sets the `needsUpdate` flag to `true`.) If neither `kARMSearch` nor `kARMSearchMore` is set, perform only a relative search. If `kARMSearch` is not set but `kARMSearchMore` is set, perform a relative search followed by an exhaustive search.

  Available in Mac OS X v10.0 and later.

  Declared in `Aliases.h`.

`kARMTryFileIDFirst`
: Perform a search using the file ID of the target before searching using the path.

  Available in Mac OS X v10.2 and later.

  Declared in `Aliases.h`.

Discussion
The `FSMatchAlias` (page 184), `FSMatchAliasNoUI` (page 187), `MatchAliasNoUI` (page 202) and `MatchAlias` (page 201) functions use these constants to specify the matching criteria by passing a sum of these constants in the *`rulesMask`* parameter. You must specify at least one of the last three parameters: `kARMSearch`, `kARMSearchMore`, and `kARMSearchRelFirst`.

Alias Resource Type

Specifies the file type of an alias resource file.

```
enum {
    rAliasType = 'alis'
};
```

Information Type Constants

The `GetAliasInfo` function uses these constants in the `index` parameter.

```
enum {
    asiZoneName = -3,
    asiServerName = -2,
    asiVolumeName = -1,
    asiAliasName = 0,
    asiParentName = 1
};
```

Constants

`asiZoneName`
If the record represents a target on an AppleShare volume, retrieve the server's zone name. Otherwise, return an empty string.

Available in Mac OS X v10.0 and later.

Declared in `Aliases.h`.

`asiServerName`
If the record represents a target on an AppleShare volume, retrieve the server name. Otherwise, return an empty string.

Available in Mac OS X v10.0 and later.

Declared in `Aliases.h`.

`asiVolumeName`
Return the name of the volume on which the target resides.

Available in Mac OS X v10.0 and later.

Declared in `Aliases.h`.

`asiAliasName`
Return the name of the target.

Available in Mac OS X v10.0 and later.

Declared in `Aliases.h`.

`asiParentName`
Return the name of the parent directory of the target of the record. If the target is a volume, return the volume name.

Available in Mac OS X v10.0 and later.

Declared in `Aliases.h`.

Gestalt Constants

You can check for version and feature availability information by using the Alias Manager selectors defined in the Gestalt Manager. For more information, see *Gestalt Manager Reference*.

Code Fragment Manager Reference

Framework: CoreServices/CoreServices.h
Declared in CodeFragments.h

Overview

This chapter describes the Code Fragment Manager, the part of the Mac OS that loads fragments into memory and prepares them for execution. A fragment can be an application, an import library, a system extension, or any other block of executable code and its associated data.

The Code Fragment Manager is intended to operate transparently to most applications and other software. You need to use the Code Fragment Manager explicitly only if

- you need to load code modules dynamically during the execution of your application or other software
- you want to unload code modules before the termination of your application
- you want to obtain information about the symbols exported by a fragment

For example, if your application supports dynamic loading of tools, filters, or other software modules contained in fragments, you'll need to use the Code Fragment Manager to load and prepare them for execution.

Carbon supports the Code Fragment Manager.

Functions by Task

Finding Symbols

`CountSymbols` (page 225) Deprecated in Mac OS X v10.5
Determines how many symbols are exported from a specified fragment.

`FindSymbol` (page 226) Deprecated in Mac OS X v10.5
Searches for a specific exported symbol.

`GetIndSymbol` (page 229) Deprecated in Mac OS X v10.5
Gets information about the exported symbols in a fragment.

Loading Fragments

`GetDiskFragment` (page 227) Deprecated in Mac OS X v10.5
Locates and possibly also loads a fragment contained in a file's data fork into your application's context.

`GetMemFragment` (page 229) Deprecated in Mac OS X v10.5
Prepares a memory-based fragment for subsequent execution.

`GetSharedLibrary` (page 231) Deprecated in Mac OS X v10.5
Locates and possibly also loads an import library into your application's context.

Unloading Fragments

`CloseConnection` (page 224) Deprecated in Mac OS X v10.5
Closes a connection to a fragment.

Converting a Bundle Prelocator

`ConvertBundlePreLocator` (page 225) Deprecated in Mac OS X v10.5
Converts a bundle prelocator to a Core Foundation bundle locator.

Functions

CloseConnection

Closes a connection to a fragment. (Deprecated in Mac OS X v10.5.)

```
OSErr CloseConnection (
   CFragConnectionID *connID
);
```

Parameters

connID
A pointer to a connection ID.

Return Value
A result code. See "Code Fragment Manager Result Codes" (page 259).

Discussion
The `CloseConnection` function closes the connection to a fragment indicated by the `connID` parameter. `CloseConnection` decrements the count of existing connections to the specified fragment and, if the resulting count is 0, calls the fragment's termination function and releases the memory occupied by the code and data sections of the fragment. If the resulting count is not 0, any per-connection data is released but the code section remains in memory.

When a fragment is unloaded as a result of its final connection having been closed, all libraries that depend on that fragment are also released, provided that their usage counts are also 0.

The Code Fragment Manager automatically closes any connections that remain open at the time `ExitToShell` is called for your application, so you need to call `CloseConnection` only for fragments you wish to unload before your application terminates.

Special Considerations

You can close a connection only to the root of a loading sequence (that is, the fragment whose loading triggered the entire load chain).

Availability
Available in CarbonLib 1.0 and later when Code Fragment Manager 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
`CodeFragments.h`

ConvertBundlePreLocator

Converts a bundle prelocator to a Core Foundation bundle locator. (Deprecated in Mac OS X v10.5.)

```
OSErr ConvertBundlePreLocator (
   CFragSystem7LocatorPtr initBlockLocator
);
```

Parameters

`initBlockLocator`
A pointer to a fragment locator structure. On input, the structure contains a System 7 locator. On output, the structure contains a `CFragCFBundleLocator`.

Return Value
A result code. See "Code Fragment Manager Result Codes" (page 259).

Discussion
This function can be used by initialization routines.

Availability
Available in Mac OS X 10.1 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`CodeFragments.h`

CountSymbols

Determines how many symbols are exported from a specified fragment. (Deprecated in Mac OS X v10.5.)

```
OSErr CountSymbols (
   CFragConnectionID connID,
   long *symCount
);
```

Parameters

connID

A connection ID.

symCount

On return, a pointer to the number of exported symbols in the fragment whose connection ID is `connID`. You can use the value returned in `symCount` to index through all the exported symbols in a particular fragment (using the `GetIndSymbol` function).

Return Value

A result code. See "Code Fragment Manager Result Codes" (page 259).

Availability

Available in CarbonLib 1.0 and later when Code Fragment Manager 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

`CodeFragments.h`

FindSymbol

Searches for a specific exported symbol. (Deprecated in Mac OS X v10.5.)

```
OSErr FindSymbol (
   CFragConnectionID connID,
   ConstStr255Param symName,
   Ptr *symAddr,
   CFragSymbolClass *symClass
);
```

Parameters

connID

A connection ID.

symName

A symbol name.

symAddr

On return, a pointer to the address of the symbol whose name is `symName`.

symClass

On return, a pointer to the class of the symbol whose name is `symName`. The currently recognized symbol classes are defined by the "Load Flag, Symbol Class, and Fragment Locator Constants" (page 252).

Return Value

A result code. See "Code Fragment Manager Result Codes" (page 259).

Discussion
The `FindSymbol` function searches the code fragment identified by the `connID` parameter for the symbol whose name is specified by the `symName` parameter. If that symbol is found, `FindSymbol` returns the address of the symbol in the `symAddr` parameter and the class of the symbol in the `symClass` parameter.

Because a fragment's code is normally exported through transition vectors to that code, the value `kCodeSymbol` is not returned in the PowerPC environment. You can use the other two constants to distinguish exports that represent code (of class `kTVectSymbol`) from those that represent general data (of class `kDataSymbol`).

Availability
Available in CarbonLib 1.0 and later when Code Fragment Manager 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
`CodeFragments.h`

GetDiskFragment

Locates and possibly also loads a fragment contained in a file's data fork into your application's context. (Deprecated in Mac OS X v10.5.)

```
OSErr GetDiskFragment (
   const FSSpec *fileSpec,
   UInt32 offset,
   UInt32 length,
   ConstStr63Param fragName,
   CFragLoadOptions options,
   CFragConnectionID *connID,
   Ptr *mainAddr,
   Str255 errMessage
);
```

Parameters

fileSpec
A pointer to a file system specification that identifies the disk-based fragment to load.

offset
The number of bytes from the beginning of the file's data fork at which the beginning of the fragment is located.

length
The length (in bytes) of the fragment. Specify the constant `kWholeFork` for this parameter if the fragment extends to the end-of-file of the data fork. Specify a nonzero value for the exact length of the fragment.

fragName
An optional name of the fragment. (This information is used primarily to allow you to identify the fragment during debugging.

loadFlags
A flag that specifies the operation to perform on the fragment.The Code Fragment Manager recognizes the constants described in "Load Flag, Symbol Class, and Fragment Locator Constants" (page 252).

connID

On return, a pointer to the connection ID that identifies the connection to the fragment. You can pass this ID to other Code Fragment Manager functions.

mainAddr

On return, a pointer to the main address of the fragment. The value returned is specific to the fragment itself. Your application can use this parameter for its own purposes.

errMessage

On return, the name of the fragment that could not successfully be loaded. This parameter is meaningful only if the call to `GetDiskFragment` fails.

Return Value

A result code. See "Code Fragment Manager Result Codes" (page 259). The `kFindLib` constant in the `loadFlags` parameter specifies that the Code Fragment Manager search for the specified fragment. If the fragment is already prepared and connected to your application, `GetDiskFragment` returns `fragNoErr`. If the specified fragment is not found, `GetDiskFragment` returns the result code `fragLibNotFound`. If the specified fragment is found but could not be connected to your application, the function returns `fragLibConnErr`.

Discussion

Loading involves finding the specified fragment, reading it into memory (if it is not already in memory), and preparing it for execution. The Code Fragment Manager attempts to resolve all symbols imported by the fragment; to do so may involve loading import libraries.

If the fragment loading fails, the Code Fragment Manager returns an error code. Note, however, that the error encountered is not always in the fragment you asked to load. Rather, the error might have occurred while attempting to load an import library that the fragment you want to load depends on. For this reason, the Code Fragment Manager also returns, in the `errMessage` parameter, the name of the fragment that caused the load to fail. Although fragment names are restricted to 63 characters, the `errMessage` parameter is declared as type `Str255`; doing this allows future versions of the Code Fragment Manager to return a more informative message in the `errMessage` parameter.

Availability

Modified in Carbon. Available in CarbonLib 1.0 and later when Code Fragment Manager 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.

Carbon Porting Notes

On Mac OS X, `GetDiskFragment` does not include the folder containing the root fragment (assuming that it is different from the application fragment) in its search path for import libraries. For example, say your application used a special folder to store plugins. If that folder also contained special libraries for those plugins, then calling `GetDiskFragment` to load a plugin would not find those libraries.

The workaround is to make sure that any import libraries you require are in the Code Fragment Manager's search path (such as by designating an application library subfolder in the code fragment resource, or placing the libraries in the application's container). For more details of how the Code Fragment Manager searches for import libraries, see Mac OS Runtime Architectures.

Declared In

`CodeFragments.h`

GetIndSymbol

Gets information about the exported symbols in a fragment. (Deprecated in Mac OS X v10.5.)

```
OSErr GetIndSymbol (
   CFragConnectionID connID,
   long symIndex,
   Str255 symName,
   Ptr *symAddr,
   CFragSymbolClass *symClass
);
```

Parameters

connID

A connection ID.

symIndex

A symbol index. This index is zero-based. That is, the value of this parameter should be between zero and the number of symbols -1 (where the number of symbols is determined by calling the `CountSymbols` (page 225) function).

symName

On return, the name of the indicated symbol.

symAddr

On return, a pointer to the address of the indicated symbol.

symClass

On return, a pointer to the class of the indicated symbol. See "Load Flag, Symbol Class, and Fragment Locator Constants" (page 252).

Return Value

A result code. See "Code Fragment Manager Result Codes" (page 259).

Discussion

If `GetIndSymbol` executes successfully, it returns the symbol's name, starting address, and class in the `symName`, `symAddr`, and `symClass` parameters, respectively. A fragment's exported symbols are retrieved in no predetermined order.

Availability

Available in CarbonLib 1.0 and later when Code Fragment Manager 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

`CodeFragments.h`

GetMemFragment

Prepares a memory-based fragment for subsequent execution. (Deprecated in Mac OS X v10.5.)

```
OSErr GetMemFragment (
   void *memAddr,
   UInt32 length,
   ConstStr63Param fragName,
   CFragLoadOptions options,
   CFragConnectionID *connID,
   Ptr *mainAddr,
   Str255 errMessage
);
```

Parameters

memAddr

The address of the fragment.

length

The size, in bytes, of the fragment.

fragName

The name of the fragment. (This information is used primarily to allow you to identify the fragment during debugging.

loadFlags

A flag that specifies the operation to perform on the fragment. The Code Fragment Manager recognizes the constants described in "Load Flag, Symbol Class, and Fragment Locator Constants" (page 252).

connID

On return, a pointer to the connection ID that identifies the connection to the fragment. You can pass this ID to other Code Fragment Manager functions (for example, `CloseConnection`).

mainAddr

On return, a pointer to the main address of the fragment. The value returned is specific to the fragment itself.

errMessage

On return, the name of the fragment that could not successfully be loaded. This parameter is meaningful only if the call to `GetMemFragment` fails.

Return Value

A result code. See "Code Fragment Manager Result Codes" (page 259).

Discussion

The `GetMemFragment` is most useful for handling code that is contained in a resource. You can read the resource data into memory using normal Resource Manager functions (for example, `Get1Resource`) and then call `GetMemFragment` to complete the processing required to prepare it for use (for example, to resolve any imports and execute the fragment's initialization function).

You must lock the resource-based fragment into memory (for example, by calling `HLock`) before calling `GetMemFragment`. You must not unlock the memory until you have closed the connection to the fragment (by calling `CloseConnection`).

Loading involves finding the specified fragment, reading it into memory (if it is not already in memory), and preparing it for execution. The Code Fragment Manager attempts to resolve all symbols imported by the fragment; to do so may involve loading import libraries.

If the fragment loading fails, the Code Fragment Manager returns an error code. Note, however, that the error encountered is not always in the fragment you asked to load. Rather, the error might have occurred while attempting to load an import library that the fragment you want to load depends on. For this reason, the Code Fragment Manager also returns, in the `errMessage` parameter, the name of the fragment that

caused the load to fail. Although fragment names are restricted to 63 characters, the `errMessage` parameter is declared as type `Str255`; doing this allows future versions of the Code Fragment Manager to return a more informative message in the `errMessage` parameter.

Availability
Available in CarbonLib 1.0 and later when Code Fragment Manager 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
`CodeFragments.h`

GetSharedLibrary

Locates and possibly also loads an import library into your application's context. (Deprecated in Mac OS X v10.5.)

```
OSErr GetSharedLibrary (
   ConstStr63Param libName,
   CFragArchitecture archType,
   CFragLoadOptions options,
   CFragConnectionID *connID,
   Ptr *mainAddr,
   Str255 errMessage
);
```

Parameters

libName
The name of an import library.

archType
The instruction set architecture of the import library. For the PowerPC architecture, use the constant `kPowerPCArch`. For the 680x0 architecture, use the constant `kMotorola68KArch`.

loadFlags
A flag that specifies the operation to perform on the import library. The Code Fragment Manager recognizes the constants described in "Load Flag, Symbol Class, and Fragment Locator Constants" (page 252).

connID
On return, a pointer to the connection ID that identifies the connection to the import library. You can pass this ID to other Code Fragment Manager functions.

mainAddr
On return, a pointer to the main address of the import library. The value returned is specific to the import library itself and is not used by the Code Fragment Manager.

errMessage
On return, the name of the fragment that could not successfully be loaded. This parameter is meaningful only if the call to `GetSharedLibrary` fails.

Return Value
A result code. See "Code Fragment Manager Result Codes" (page 259).

Discussion
The `GetSharedLibrary` function locates the import library named by the `libName` parameter and possibly also loads that import library into your application's context. The actions of `GetSharedLibrary` depend on the action flag you pass in the `loadFlags` parameter; pass `kFindLib` to get the connection ID of an existing connection to the specified fragment, `kLoadLib` to load the specified fragment, or `kLoadNewCopy` to load the fragment with a new copy of the fragment's data section.

The `GetSharedLibrary` function does not resolve any unresolved imports in your application. In particular, you cannot use it to resolve any weak imports in your code fragment.

Loading involves finding the specified fragment, reading it into memory (if it is not already in memory), and preparing it for execution. The Code Fragment Manager attempts to resolve all symbols imported by the fragment; to do so may involve loading import libraries.

If the fragment loading fails, the Code Fragment Manager returns an error code. Note, however, that the error encountered is not always in the fragment you asked to load. Rather, the error might have occurred while attempting to load an import library that the fragment you want to load depends on. For this reason, the Code Fragment Manager also returns, in the `errMessage` parameter, the name of the fragment that caused the load to fail. Although fragment names are restricted to 63 characters, the `errMessage` parameter is declared as type `Str255`; doing this allows future versions of the Code Fragment Manager to return a more informative message in the `errMessage` parameter.

Availability
Available in CarbonLib 1.0 and later when Code Fragment Manager 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
`CodeFragments.h`

Callbacks

CFragInitFunction

Defines a fragment initialization function that is executed by the Code Fragment Manager when the fragment is first loaded into memory and prepared for execution.

```
typedef OSErr (*CFragInitFunction) (
    const CFragInitBlock * initBlock
);
```

If you name your function `MyCFragInitFunction`, you would declare it like this:

```
OSErr MyCFragInitFunction (
    const CFragInitBlock * initBlock
);
```

Parameters
`initBlock`
A pointer to a fragment initialization block specifying information about the fragment.

Return Value
A result code. See "Code Fragment Manager Result Codes" (page 259). Your initialization function should return `noErr` if it executes successfully, and some other result code if it does not. If your initialization function returns any result code other than `noErr`, the entire load fails and the error `fragUserInitProcErr` is returned to the code that requested the root load.

Discussion
A fragment's initialization function is executed immediately before the fragment's main function (if it has one) is executed. The initialization function is passed a pointer to an initialization block, which contains information about the fragment, such as its location and connection ID. See `InitBlock` (page 245) for a description of the fields of the initialization block.

You can use the initialization function to perform any tasks that need to be performed before any of the code or data in the fragment is accessed. For example, you might want to open the fragment's resource fork (if it has one). You can determine the location of the fragment's container from the `FragmentLocator` field of the fragment initialization block whose address is passed to your initialization function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CodeFragments.h`

CFragTermProcedure

Defines a pointer to a fragment termination function that is executed by the Code Fragment Manager when the fragment is unloaded from memory.

```
typedef void (*CFragTermProcedure) (
);
```

If you name your function `MyCFragTermProcedure`, you would declare it like this:

```
void MyCFragTermProcedure ();
```

Discussion
A fragment's termination function is executed immediately before the fragment is unloaded from memory. You can use the termination function to perform any necessary clean-up tasks, such as closing open resource files or disposing of any memory allocated by the fragment.

Note that a termination function is not passed any parameters and does not return any result. You are expected to maintain any information about the fragment (such as file reference numbers of any open files) in its static data area.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CodeFragments.h`

Data Types

CFragCFBundleLocator

```
struct CFragCFBundleLocator {
    CFBundleRef fragmentBundle;
    UInt32 offset;
    UInt32 length;
};
typedef struct CFragCFBundleLocator CFragCFBundleLocator;
```

Fields

`fragmentBundle`

`offset`

`length`

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CodeFragments.h`

CFragClosureID

```
typedef struct OpaqueCFragClosureID * CFragClosureID;
```

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CodeFragments.h`

CFragConnectionID

```
typedef struct OpaqueCFragConnectionID * CFragConnectionID;
```

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CodeFragments.h`

CFragContainerID

```
typedef struct OpaqueCFragContainerID * CFragContainerID;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CodeFragments.h`

CFragContextID

```
typedef MPProcessID CFragContextID;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CodeFragments.h`

CFragHFSDiskFlatLocator

```
typedef CFragSystem7DiskFlatLocator CFragHFSDiskFlatLocator;
```

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

Declared In
`CodeFragments.h`

CFragHFSLocator

```
typedef CFragSystem7Locator CFragHFSLocator;
```

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

Declared In
`CodeFragments.h`

CFragHFSLocatorPtr

```
typedef CFragSystem7LocatorPtr CFragHFSLocatorPtr;
```

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

Declared In
CodeFragments.h

CFragHFSMemoryLocator

```
typedef CFragSystem7MemoryLocator CFragHFSMemoryLocator;
```

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

Declared In
CodeFragments.h

CFragHFSSegmentedLocator

```
typedef CFragSystem7SegmentedLocator CFragHFSSegmentedLocator;
```

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

Declared In
CodeFragments.h

CFragInitBlock

```
typedef CFragSystem7InitBlock CFragInitBlock;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
CodeFragments.h

CFragInitBlockPtr

```
typedef CFragSystem7InitBlockPtr CFragInitBlockPtr;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
CodeFragments.h

CFragResource

```
struct CFragResource {
    UInt32 reservedA;
    UInt32 reservedB;
    UInt16 reservedC;
    UInt16 version;
    UInt32 reservedD;
    UInt32 reservedE;
    UInt32 reservedF;
    UInt32 reservedG;
    UInt16 reservedH;
    UInt16 memberCount;
    CFragResourceMember firstMember;
};
typedef struct CFragResource CFragResource;
typedef CFragResource * CFragResourcePtr;
typedef CFragResourcePtr * CFragResourceHandle;
```

Fields

`reservedA`
This field is reserved for future use. Set this field to 0.

`reservedB`
This field is reserved for future use. Set this field to 0.

`reservedC`
This field is reserved for future use. Set this field to 0.

`version`

`reservedD`
This field is reserved for future use. Set this field to 0.

`reservedE`
This field is reserved for future use. Set this field to 0.

`reservedF`
This field is reserved for future use. Set this field to 0.

`reservedG`
This field is reserved for future use. Set this field to 0.

`reservedH`
This field is reserved for future use. Set this field to 0.

`memberCount`

`firstMember`

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CodeFragments.h`

CFragResourceExtensionHeader

```
struct CFragResourceExtensionHeader {
    UInt16 extensionKind;
    UInt16 extensionSize;
};
typedef struct CFragResourceExtensionHeader CFragResourceExtensionHeader;
typedef CFragResourceExtensionHeader * CFragResourceExtensionHeaderPtr;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CodeFragments.h`

CFragResourceMember

```
struct CFragResourceMember {
    CFragArchitecture architecture;
    UInt16 reservedA;
    UInt8 reservedB;
    UInt8 updateLevel;
    CFragVersionNumber currentVersion;
    CFragVersionNumber oldDefVersion;
    CFragUsage1Union uUsage1;
    CFragUsage2Union uUsage2;
    CFragUsage usage;
    CFragLocatorKind where;
    UInt32 offset;
    UInt32 length;
    CFragWhere1Union uWhere1;
    CFragWhere2Union uWhere2;
    UInt16 extensionCount;
    UInt16 memberSize;
    unsigned char name[16];
};
typedef struct CFragResourceMember CFragResourceMember;
typedef CFragResourceMember * CFragResourceMemberPtr;
```

Fields
`architecture`
`reservedA`
This field is reserved. Set to 0.
`reservedB`
This field is reserved. Set to 0.

`updateLevel`
`currentVersion`
`oldDefVersion`
`uUsage1`
`uUsage2`
`usage`
`where`
`offset`
`length`
`uWhere1`
`uWhere2`
`extensionCount`
Specifies the number of extensions beyond the name.

`memberSize`
Specifies the size in bytes, including all extensions.

`name`

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CodeFragments.h`

CFragResourceSearchExtension

```
struct CFragResourceSearchExtension {
    CFragResourceExtensionHeader header;
    OSType libKind;
    unsigned char qualifiers[1];
};
typedef struct CFragResourceSearchExtension CFragResourceSearchExtension;
typedef CFragResourceSearchExtension * CFragResourceSearchExtensionPtr;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CodeFragments.h`

CFragSystem7DiskFlatLocator

Defines a disk location structure.

```
struct CFragSystem7DiskFlatLocator {
    FSSpec * fileSpec;
    UInt32 offset;
    UInt32 length;
};
typedef struct CFragSystem7DiskFlatLocator CFragSystem7DiskFlatLocator;
typedef CFragSystem7DiskFlatLocator DiskFragment;
```

Fields

`fileSpec`
: A pointer to a file specification structure (a data structure of type `FSSpec`) for the data fork of a file. This pointer is valid only while the initialization function is executing. If you need to access the information in the file specification structure at any later time, you must make a copy of that structure.

`offset`
: The offset, in bytes, from the beginning of the file's data fork to the beginning of the fragment.

`length`
: The length, in bytes, of the fragment. If this field contains the value 0, the fragment extends to the end-of-file.

Discussion

For fragments located in the data fork of a file on disk, the `onDisk` field of a fragment location structure contains a disk location structure, which specifies the location of the fragment.

The fields of a fragment initialization block are aligned in memory in accordance with 680x0 alignment conventions.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CodeFragments.h`

CFragSystem7InitBlock

```
struct CFragSystem7InitBlock {
    CFragContextID contextID;
    CFragClosureID closureID;
    CFragConnectionID connectionID;
    CFragSystem7Locator fragLocator;
    StringPtr libName;
    UInt32 reservedA;
};
typedef struct CFragSystem7InitBlock CFragSystem7InitBlock;
typedef CFragSystem7InitBlock * CFragSystem7InitBlock;
typedef CFragSystem7InitBlock CFragInitBlock;
```

Fields

`contextID`
: A context ID.

`closureID`
: A closure ID.

`connectionID`
A connection ID.

`fragLocator`
A fragment location structure, `CFragSystem7Locator` (page 241) that specifies the location of the fragment.

`libName`
A pointer to the name of the fragment being initialized. The name is a Pascal string (a length byte followed by the name itself).

`reservedA`
Reserved for use by Apple Computer.

Discussion
The Code Fragment Manager passes to your fragment's initialization function a pointer to a fragment initialization block, which contains information about the fragment. A fragment initialization block is defined by the `InitBlock` data type.

The fields of a fragment initialization block are aligned in memory in accordance with 680x0 alignment conventions.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CodeFragments.h`

CFragSystem7Locator

Defines a fragment location structure.

```
struct CFragSystem7Locator {
    SInt32 where
    union {
        CFragSystem7DiskFlatLocator onDisk;
        CFragSystem7MemoryLocator inMem;
        CFragSystem7SegmentedLocator inSegs;
        CFragCFBundleLocator inBundle;
    } u;
};
typedef struct CFragSystem7Locator CFragSystem7Locator;
typedef CFragSystem7Locator * CFragSystem7LocatorPtr;
typedef CFragSystem7Locator FragmentLocator;
```

Fields
`where`
A selector that determines which member of the following union is relevant. This field can contain one of the constants described in "Load Flag, Symbol Class, and Fragment Locator Constants" (page 252).

`u`
If the `where` field has the value `kOnDiskFlat`, a disk location structure.

Discussion
The `fragLocator` field of an initialization block contains a fragment location structure that provides information about the location of a fragment.

The fields of a fragment initialization block are aligned in memory in accordance with 680x0 alignment conventions.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CodeFragments.h`

CFragSystem7MemoryLocator

Defines a memory location structure.

```
struct CFragSystem7MemoryLocator {
    LogicalAddress address;
    UInt32 length;
    Boolean inPlace;
    UInt8 reservedA;
    UInt16 reservedB;
};
typedef struct CFragSystem7MemoryLocator CFragSystem7MemoryLocator;
typedef CFragSystem7MemoryLocator MemFragment;
```

Fields

`address`
: A pointer to the beginning of the fragment in memory.

`length`
: The length, in bytes, of the fragment.

`inPlace`
: A Boolean value that specifies whether the container's data section is instantiated in place (`true`) or elsewhere (`false`).

`reservedA`
: This field is reserved for future use. Set to 0.

`reservedB`
: This field is reserved for future use. Set to 0.

Discussion
For fragments located in memory, the `inMem` field of a fragment location structure contains a memory location structure, which specifies the location of the fragment in memory.

The fields of a fragment initialization block are aligned in memory in accordance with 680x0 alignment conventions.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CodeFragments.h`

CFragSystem7SegmentedLocator

Defines a segment location structure.

```
struct CFragSystem7SegmentedLocator {
    FSSpec * fileSpec;
    OSType rsrcType;
    SInt16 rsrcID;
    UInt16 reservedA;
};
typedef struct CFragSystem7SegmentedLocator CFragSystem7SegmentedLocator;
typedef CFragSystem7SegmentedLocator SegmentedFragment;
```

Fields

`fileSpec`
: A pointer to a file specification structure (a data structure of type `FSSpec`) for the resource fork of a file. This pointer is valid only while the initialization function is executing. If you need to access the information in the file specification structure at any later time, you must make a copy of that structure.

`rsrcType`
: The resource type of the resource containing the fragment.

`rsrcID`
: The resource ID of the resource containing the fragment.

`reservedA`
: This field is reserved for future use.

Discussion

For fragments located in the resource fork of a file on disk, the `inSegs` field of a fragment location structure contains a segment location structure, which specifies the location of the fragment.

The fields of a fragment initialization block are aligned in memory in accordance with 680x0 alignment conventions.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CodeFragments.h`

CFragUsage1Union

```
union CFragUsage1Union {
    UInt32 appStackSize;
};
typedef union CFragUsage1Union CFragUsage1Union;
```

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CodeFragments.h`

CFragUsage2Union

```
union CFragUsage2Union {
    SInt16 appSubdirID;
    UInt16 libFlags;
};
typedef union CFragUsage2Union CFragUsage2Union;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CodeFragments.h`

CFragWhere1Union

```
union CFragWhere1Union {
    UInt32 spaceID;
};
typedef union CFragWhere1Union CFragWhere1Union;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CodeFragments.h`

CFragWhere2Union

```
union CFragWhere2Union {
    UInt16 reserved;
};
typedef union CFragWhere2Union CFragWhere2Union;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CodeFragments.h`

ConnectionID

```
typedef CFragConnectionID ConnectionID;
```

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

Declared In
CodeFragments.h

DiskFragment

A CFragSystem7DiskFlatLocator structure.

```
typedef CFragSystem7DiskFlatLocator DiskFragment;
```

Discussion
See CFragSystem7DiskFlatLocator (page 239).

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

Declared In
CodeFragments.h

FragmentLocator

A CFragSystem7Locator structure.

```
typedef CFragSystem7Locator FragmentLocator;
```

Discussion
See CFragSystem7Locator (page 241).

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

Declared In
CodeFragments.h

FragmentLocatorPtr

```
typedef CFragSystem7LocatorPtr FragmentLocatorPtr;
```

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

Declared In
CodeFragments.h

InitBlock

```
typedef CFragInitBlock InitBlock;
```

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

Declared In
CodeFragments.h

InitBlockPtr

```
typedef CFragInitBlockPtr InitBlockPtr;
```

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

Declared In
CodeFragments.h

LoadFlags

```
typedef CFragLoadOptions LoadFlags;
```

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

Declared In
CodeFragments.h

MemFragment

A `CFragSystem7MemoryLocator` structure.

```
typedef CFragSystem7MemoryLocator MemFragment;
```

Discussion
See `CFragSystem7MemoryLocator` (page 242).

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

Declared In
CodeFragments.h

SegmentedFragment

A `CFragSystem7SegmentedLocator` structure.

```
typedef CFragSystem7SegmentedLocator SegmentedFragment;
```

Discussion
See `CFragSystem7SegmentedLocator` (page 243).

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

Declared In
`CodeFragments.h`

SymClass

```
typedef CFragSymbolClass SymClass;
```

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

Declared In
`CodeFragments.h`

Constants

Architecture Constants

```
typedef OSType CFragArchitecture;
enum {
    kPowerPCCFragArch = 'pwpc',
    kMotorola68KCFragArch = 'm68k',
    kAnyCFragArch = 0x3F3F3F3F
};
```

Constants

`kPowerPCCFragArch`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kMotorola68KCFragArch`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kAnyCFragArch`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

Code Fragment Kind

```
enum {
    kIsCompleteCFrag = 0,
    kFirstCFragUpdate = 1
};
```

Constants

`kIsCompleteCFrag`

Indicates a base fragment rather than an update.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CodeFragments.h`.

`kFirstCFragUpdate`

Indicates the first update, others are numbered starting with 2.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CodeFragments.h`.

Current Resource Version

```
enum {
    kCurrCFragResourceVersion = 1
};
```

Default Name Length

```
enum {
    kDefaultCFragNameLen = 16
};
```

File Location

```
#define IsFileLocation CFragHasFileLocation;
```

kCFragGoesToEOF

```
enum {
    kCFragGoesToEOF = 0
};
```

kCFragLibUsageMapPrivatelyMask

```
enum {
    kCFragLibUsageMapPrivatelyMask = 0x0001
};
```

Constants

`kCFragLibUsageMapPrivatelyMask`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CodeFragments.h`.

kCFragResourceSearchExtensionKind

```
enum {
    kCFragResourceSearchExtensionKind = 0x30EE
};
```

kCFragResourceType

```
enum {
    kCFragResourceType = 'cfrg',
    kCFragResourceID = 0,
    kCFragLibraryFileType = 'shlb',
    kCFragAllFileTypes = 0xFFFFFFFF
};
```

Constants

`kCFragResourceType`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kCFragResourceID`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kCFragLibraryFileType`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kCFragAllFileTypes`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

kCompiledCFragArch

```
enum {
    kCompiledCFragArch = 'kPowerPCCFragArch'
};
```

Constants

`kCompiledCFragArch`
The value for this constant is `'kPowerPCCFragArch'` if you have defined `TARGET_CPU_PPC`. If you define `TARGET_CPU_X86`, then the value of this constant is `'none'`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

kLoadCFrag

```
enum {
    kLoadCFrag = kReferenceCFrag
};
```

Constants

`kLoadCFrag`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CodeFragments.h`.

kPowerPC

```
enum {
    kPowerPC = kPowerPCCFragArch,
    kMotorola68K = kMotorola68KCFragArch
};
```

Constants

`kPowerPC`

Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kMotorola68K`

Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

Load Flag, Symbol Class, and Fragment Locator Constants

```
enum {
    kPowerPCArch = kPowerPCCFragArch,
    kMotorola68KArch = kMotorola68KCFragArch,
    kAnyArchType = kAnyCFragArch,
    kNoLibName = 0,
    kNoConnectionID = 0,
    kLoadLib = kLoadCFrag,
    kFindLib = kFindCFrag,
    kNewCFragCopy = kPrivateCFragCopy,
    kLoadNewCopy = kPrivateCFragCopy,
    kUseInPlace = 0x80,
    kCodeSym = kCodeCFragSymbol,
    kDataSym = kDataCFragSymbol,
    kTVectSym = kTVectorCFragSymbol,
    kTOCSym = kTOCCFragSymbol,
    kGlueSym = kGlueCFragSymbol,
    kInMem = kMemoryCFragLocator,
    kOnDiskFlat = kDataForkCFragLocator,
    kOnDiskSegmented = kResourceCFragLocator,
    kIsLib = kImportLibraryCFrag,
    kIsApp = kApplicationCFrag,
    kIsDropIn = kDropInAdditionCFrag,
    kFullLib = kIsCompleteCFrag,
    kUpdateLib = kFirstCFragUpdate,
    kWholeFork = kCFragGoesToEOF,
    kCFMRsrcType = kCFragResourceType,
    kCFMRsrcID = kCFragResourceID,
    kSHLBFileType = kCFragLibraryFileType,
    kUnresolvedSymbolAddress = kUnresolvedCFragSymbolAddress
};
```

Constants

`kPowerPCArch`

Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kMotorola68KArch`

Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kAnyArchType`

Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kNoLibName`

Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kNoConnectionID`

Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kLoadLib`
Specifies that the Code Fragment Manager search for the specified fragment.

Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kFindLib`
Specifies that the Code Fragment Manager search for the specified fragment and, if it finds it, load it into memory. If the fragment has already been loaded, it is not loaded again. The Code Fragment Manager uses the data-instantiation method specified in the fragment's container (which is either global or per-connection instantiation).

Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kNewCFragCopy`
Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kLoadNewCopy`
Specifies that the Code Fragment Manager load the specified fragment, creating a new copy of any writable data maintained by the fragment. You specify `kLoadNewCopy` to obtain one instance per load of the fragment's data and to override the data-instantiation method specified in the container itself. This is most useful for application extensions (for example, drop-in tools).

Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kUseInPlace`
Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kCodeSym`
Specifies a code symbol.

Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kDataSym`
Specifies a data symbol.

Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kTVectSym`
Specifies a transition vector symbol.

Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kTOCSym`
Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kGlueSym`
Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kInMem`
Specifies that the container is in memory. If used in the `where` parameter of a `FragmentLocator` structure, the relevant member of the union is a `CFragSystem7SegmentedLocator` (page 243) structure.

Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kOnDiskFlat`
Specifies that the container is in a data fork. If used in the `where` parameter of a `FragmentLocator` structure, the relevant member of the union is a `CFragSystem7SegmentedLocator` (page 243) structure.

Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kOnDiskSegmented`
Specifies that the container is in a resource. If used in the `where` parameter of a `FragmentLocator` structure, the relevant member of the union is a `CFragSystem7SegmentedLocator` (page 243) structure.

Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kIsLib`
Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kIsApp`
Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kIsDropIn`
Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kFullLib`
Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kUpdateLib`
Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kWholeFork`
Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kCFMRsrcType`
Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kCFMRsrcID`
Available in Mac OS X v10.0 and later.

Declared in `CodeFragments.h`.

`kSHLBFileType`
Available in Mac OS X v10.0 and later.
Declared in `CodeFragments.h`.

`kUnresolvedSymbolAddress`
Available in Mac OS X v10.0 and later.
Declared in `CodeFragments.h`.

Discussion
The load flag constants (`kLoadLib`, `kFindLib`, and `kLoadNewCopy`) are used in the `loadFlags` parameter of the `GetDiskFragment` (page 227), `GetMemFragment` (page 229), and `GetSharedLibrary` (page 231) functions to specify the action taken by those functions.

The symbol class constants (`kCodeSym`, `kDataSym`, and `kTVectSym`) are returned in the `symClass` parameter of the `FindSymbol` (page 226) function to specify the class of the specified symbol.

The fragment locator constants (`kInMem`, `kOnDiskFlat`, and `kOnDiskSegmented`) are used in the where field of the `FragmentLocator` (page 245) structure to indicate which member of the union `u` is relevant.

Load Options

```
typedef OptionBits CFragLoadOptions;
enum {
    kReferenceCFrag = 0x0001,
    kFindCFrag = 0x0002,
    kPrivateCFragCopy = 0x0005
};
```

Constants
`kReferenceCFrag`
Try to use existing copy, increment reference counts.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kFindCFrag`
Try find an existing copy, do not increment reference counts.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kPrivateCFragCopy`
Prepare a new private copy.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

Locator Kind

```
typedef UInt8 CFragLocatorKind;
enum {
    kMemoryCFragLocator = 0,
    kDataForkCFragLocator = 1,
    kResourceCFragLocator = 2,
    kNamedFragmentCFragLocator = 4,
    kCFBundleCFragLocator = 5,
    kCFBundlePreCFragLocator = 6
};
```

Constants

`kMemoryCFragLocator`
Indicates the container is in memory.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kDataForkCFragLocator`
Indicates the container is in a file's data fork.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kResourceCFragLocator`
Indicates the container is in a file's resource fork.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kNamedFragmentCFragLocator`
This constant is reserved for future use.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kCFBundleCFragLocator`
Indicates the container is in the executable of a `CFBundle`.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kCFBundlePreCFragLocator`
Indicates it was passed to the initialization routines in lieu of `kCFBundleCFragLocator`
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

Symbol Class Constants

```
typedef UInt8 CFragSymbolClass;
enum {
    kCodeCFragSymbol = 0,
    kDataCFragSymbol = 1,
    kTVectorCFragSymbol = 2,
    kTOCCFragSymbol = 3,
    kGlueCFragSymbol = 4
};
```

Constants

`kCodeCFragSymbol`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kDataCFragSymbol`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kTVectorCFragSymbol`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kTOCCFragSymbol`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kGlueCFragSymbol`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

Unresolved Symbol Address

```
enum {
    kUnresolvedCFragSymbolAddress = 0
};
```

Constants

`kUnresolvedCFragSymbolAddress`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

Usage Constants

```
typedef UInt8 CFragUsage;
enum {
    kImportLibraryCFrag = 0,
    kApplicationCFrag = 1,
    kDropInAdditionCFrag = 2,
    kStubLibraryCFrag = 3,
    kWeakStubLibraryCFrag = 4
};
```

Constants

`kImportLibraryCFrag`
Indicates a standard CFM import library.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kApplicationCFrag`
Indicates a MacOS application.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kDropInAdditionCFrag`
Indicates an application or library private extension/plug-in.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kStubLibraryCFrag`
Indicates an import library used for linking only
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

`kWeakStubLibraryCFrag`
Indicates an import library used for linking only and will be automatically weak linked
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `CodeFragments.h`.

Version Number

```
typedef UInt32 CFragVersionNumber;
enum {
    kNullCFragVersion = 0,
    kWildcardCFragVersion = 0xFFFFFFFF
};
```

Constants

`kNullCFragVersion`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CodeFragments.h`.

`kWildcardCFragVersion`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CodeFragments.h`.

Result Codes

The most common result codes returned by Code Fragment Manager are listed in the table below. The Code Fragment Manager may also return `paramErr` (-50).

Result Code	Value	Description
`cfragContextIDErr`	-2800	The context ID was not valid. Available in Mac OS X v10.0 and later.
`cfragFirstErrCode`	-2800	The first value in the range of CFM errors. Available in Mac OS X v10.0 and later.
`cfragConnectionIDErr`	-2801	The connection ID was not valid. Available in Mac OS X v10.0 and later.
`cfragNoSymbolErr`	-2802	The specified symbol was not found. Available in Mac OS X v10.0 and later.
`cfragNoSectionErr`	-2803	The specified section was not found. Available in Mac OS X v10.0 and later.
`cfragNoLibraryErr`	-2804	The named library was not found. Available in Mac OS X v10.0 and later.
`cfragDupRegistrationErr`	-2805	The registration name was already in use. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
cfragFragmentFormatErr	-2806	A fragment's container format is unknown. Available in Mac OS X v10.0 and later.
cfragUnresolvedErr	-2807	A fragment had "hard" unresolved imports. Available in Mac OS X v10.0 and later.
cfragNoPositionErr	-2808	The registration insertion point was not found. Available in Mac OS X v10.0 and later.
cfragNoPrivateMemErr	-2809	Out of memory for internal bookkeeping. Available in Mac OS X v10.0 and later.
cfragNoClientMemErr	-2810	Out of memory for fragment mapping or section instances. Available in Mac OS X v10.0 and later.
cfragNoIDsErr	-2811	No more CFM IDs for contexts, connections, etc. Available in Mac OS X v10.0 and later.
cfragInitOrderErr	-2812	Available in Mac OS X v10.0 and later.
cfragImportTooOldErr	-2813	An import library was too old for a client. Available in Mac OS X v10.0 and later.
cfragImportTooNewErr	-2814	An import library was too new for a client. Available in Mac OS X v10.0 and later.
cfragInitLoopErr	-2815	Circularity in required initialization order. Available in Mac OS X v10.0 and later.
cfragInitAtBootErr	-2816	A boot library has an initialization function. (System 7 only) Available in Mac OS X v10.0 and later.
cfragCFMStartupErr	-2818	Internal error during CFM initialization. Available in Mac OS X v10.0 and later.
cfragCFMInternalErr	-2819	An internal inconsistency has been detected. Available in Mac OS X v10.0 and later.
cfragFragmentCorruptErr	-2820	A fragment's container was corrupt (known format). Available in Mac OS X v10.0 and later.
cfragInitFunctionErr	-2821	A fragment's initialization routine returned an error. Available in Mac OS X v10.0 and later.
cfragNoApplicationErr	-2822	No application member found in the cfrg resource. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
cfragArchitectureErr	-2823	A fragment has an unacceptable architecture. Available in Mac OS X v10.0 and later.
cfragFragmentUsageErr	-2824	A semantic error in usage of the fragment. Available in Mac OS X v10.0 and later.
cfragFileSizeErr	-2825	A file was too large to be mapped. Available in Mac OS X v10.0 and later.
cfragNotClosureErr	-2826	The closure ID was actually a connection ID. Available in Mac OS X v10.0 and later.
cfragNoRegistrationErr	-2827	The registration name was not found. Available in Mac OS X v10.0 and later.
cfragContainerIDErr	-2828	The fragment container ID was not valid. Available in Mac OS X v10.0 and later.
cfragClosureIDErr	-2829	The closure ID was not valid. Available in Mac OS X v10.0 and later.
cfragAbortClosureErr	-2830	Used by notification handlers to abort a closure. Available in Mac OS X v10.0 and later.
cfragOutputLengthErr	-2831	An output parameter is too small to hold the value. Available in Mac OS X v10.0 and later.
cfragMapFileErr	-2851	A file could not be mapped. Available in Mac OS X v10.4 and later.
cfragExecFileRefErr	-2854	Bundle does not have valid executable file. Available in Mac OS X v10.4 and later.
cfragStdFolderErr	-2855	Could not find standard CFM folder. Available in Mac OS X v10.4 and later.
cfragRsrcForkErr	-2856	Resource fork could not be opened. Available in Mac OS X v10.4 and later.
cfragCFragRsrcErr	-2857	'cfrg' resource could not be loaded. Available in Mac OS X v10.4 and later.
cfragFirstReservedCode	-2897	Reserved value for internal warnings. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`cfragReservedCode_3`	-2897	Reserved value for internal warnings. Available in Mac OS X v10.0 and later.
`cfragReservedCode_2`	-2898	Reserved value for internal warnings. Available in Mac OS X v10.0 and later.
`cfragLastErrCode`	-2899	The last value in the range of CFM errors. Available in Mac OS X v10.0 and later.
`cfragReservedCode_1`	-2899	Available in Mac OS X v10.0 and later.

Collection Manager Reference

Framework: CoreServices/CoreServices.h

Declared in Collections.h

Overview

The Collection Manager implements an abstract data type that allows you to store multiple pieces of related information. This abstract data type is called a collection object. A collection object, or simply a collection, is an abstract data type that allows you to store information.

A collection is like an array in that it contains a number of individually accessible items. However, a collection offers some advantages over an array:

- A collection allows for a variable number of data items. You can add items to a collection or remove items from a collection during run time, and the Collection Manager automatically resizes the collection.
- A collection allows for variable-size items. Each item in a collection can contain data of any size.

A collection is also similar to a database, in that you can store information and retrieve it using a variety of search mechanisms.

The internal structure of a collection object is private–you must store information in a collection and retrieve information from it by providing a Collection Manager function with a reference to the collection. You use the functions provided by the Collection Manager to

- create and manipulate collection objects
- add information to a collection object
- retrieve information from a collection object
- store a collection object to disk and retrieve a collection object from disk

Carbon fully supports the Collection Manager.

Functions by Task

Adding and Replacing Items in a Collection

`AddCollectionItem` (page 267)
Adds a new item to a collection or to replace an existing item in a collection.

`ReplaceIndexedCollectionItem` (page 295)
Replaces the variable-length data of an item in a collection given the item's index.

Cloning and Copying Collection Objects

`CloneCollection` (page 270)
Clones a collection object—that is, increment its owner count.

`CopyCollection` (page 271)
Creates a copy of an existing collection.

`CountCollectionOwners` (page 272)
Determines the number of existing references to a collection object.

Counting Items in a Collection

`CountCollectionItems` (page 272)
Determines the total number of items in a collection.

`CountTaggedCollectionItems` (page 273)
Obtains the total number of items in a collection that have a specified collection tag.

Creating and Disposing of Collection Objects

`DisposeCollection` (page 274)
Disposes of a collection object.

`NewCollection` (page 291)
Creates a new, empty collection object.

Editing Item Attributes

`SetCollectionItemInfo` (page 299)
Edits the attributes of a specific collection item given the item's collection tag and collection ID.

`SetIndexedCollectionItemInfo` (page 300)
Edits the attributes of a specific collection item given the item's collection index.

Flattening and Unflattening a Collection

`FlattenCollection` (page 275)
Converts a collection object into a stream format suitable for storing and unflattening.

`FlattenPartialCollection` (page 277)
Converts a collection object into a stream format suitable for storage and unflattening.

`UnflattenCollection` (page 301)
Unflattens a collection that was flattened using the `FlattenCollection` or `FlattenPartialCollection` function.

Getting and Setting the Default Attributes for a Collection

`GetCollectionDefaultAttributes` (page 278)
Examines the default attributes of a collection object.

`SetCollectionDefaultAttributes` (page 298)
Alters the default attributes of a collection object.

Getting and Setting the Exception Procedure for a Collection

`GetCollectionExceptionProc` (page 279)
Obtains a pointer to the exception procedure installed in a specified collection.

`SetCollectionExceptionProc` (page 299)
Installs an exception procedure in a collection object.

Getting Information About a Collection Item

`GetCollectionItemInfo` (page 281)
Obtains information about a specific collection item given the item's collection tag and collection ID.

`GetIndexedCollectionItemInfo` (page 285)
Obtains information about a specific collection item given the item's collection index.

`GetTaggedCollectionItemInfo` (page 288)
Obtains information about a specific collection item given the item's collection tag and tag list position.

Getting Information About Collection Tags

`CollectionTagExists` (page 270)
Determines whether any of the items in a specified collection contain a specified collection tag.

`CountCollectionTags` (page 273)
Determines the number of distinct collection tags contained by the items of a specified collection.

`GetIndexedCollectionTag` (page 286)
Examines a specific collection tag contained in a collection.

Reading Collections From Resource Files

`GetNewCollection` (page 287)
Reads a collection in from a collection (`'cltn'`) resource.

Removing Items From a Collection

`EmptyCollection` (page 275)
Removes every item in a collection.

`PurgeCollection` (page 292)
Removes all items in a collection whose attributes match a specified pattern.

`PurgeCollectionTag` (page 293)
Removes all items with a specific collection tag from a collection.

`RemoveCollectionItem` (page 294)
Removes an item from a collection given the item's associated collection tag and collection ID.

`RemoveIndexedCollectionItem` (page 295)
Removes an item from a collection given the item's index.

Retrieving the Variable-Length Data From an Item

`GetCollectionItem` (page 280)
Obtains a copy of the variable-length data associated with a collection item given the item's collection tag and collection ID.

`GetIndexedCollectionItem` (page 283)
Obtains a copy of the variable-length data associated with a collection item given the item's collection index.

`GetTaggedCollectionItem` (page 287)
Obtains a copy of the variable-length data associated with a collection item given the item's collection tag and tag list position.

Working With Macintosh Memory Manager Handles

`AddCollectionItemHdl` (page 268)
Adds a new item to a collection or to replace an existing item in a collection, specifying the item's variable-length data using a handle rather than a pointer and a data size.

`FlattenCollectionToHdl` (page 276)
Flattens a collection into a Macintosh Memory Manager handle.

`GetCollectionItemHdl` (page 281)
Obtains a copy of the variable-length data associated with a collection item given the item's collection tag and collection ID.

`GetIndexedCollectionItemHdl` (page 284)
Copies the variable-length data associated with a collection item into a Macintosh Memory Manager handle, given the item's collection index.

`ReplaceIndexedCollectionItemHdl` (page 297)
Replaces the variable-length data of an item in a collection given the item's collection index, specifying the item's new variable-length data using a handle rather than a pointer and a data size.

`UnflattenCollectionFromHdl` (page 302)
Unflattens a collection that was flattened using the `FlattenCollectionToHdl` utility function.

Working With Universal Procedure Pointers

`NewCollectionExceptionUPP` (page 291)
Creates a new universal procedure pointer (UPP) to an error-handling callback.

`InvokeCollectionExceptionUPP` (page 290)
Calls an error-handling callback.

`DisposeCollectionExceptionUPP` (page 274)
Disposes of a universal procedure pointer (UPP) to an error-handling callback.

`NewCollectionFlattenUPP` (page 292)
Creates a new universal procedure pointer (UPP) to a data-flattening callback.

`InvokeCollectionFlattenUPP` (page 290)
Calls a data-flattening callback.

`DisposeCollectionFlattenUPP` (page 275)
Disposes of a universal procedure pointer (UPP) to a data-flattening callback.

Retaining And Releasing

`RetainCollection` (page 298)
Increments the owner count (the number of existing references) for a collection object.

`GetCollectionRetainCount` (page 283)
Obtains the owner count (the number of existing references) for a collection object.

`ReleaseCollection` (page 294)
Decrements the owner count (the number of existing references) for a collection object.

Functions

AddCollectionItem

Adds a new item to a collection or to replace an existing item in a collection.

```
OSErr AddCollectionItem (
   Collection c,
   CollectionTag tag,
   SInt32 id,
   SInt32 itemSize,
   const void *itemData
);
```

Parameters

c
A reference to the collection you want to add the item to. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag
The collection tag you want to associate with the new item.

id
The collection ID you want to associate with the new item.

itemSize
The size in bytes of the item's variable-length data.

itemData
A pointer to the item's variable-length data.

Return Value
A result code. See "Result Codes" (page 313).

Discussion
The `AddCollectionItem` function adds an item to the collection referenced by the `c` parameter. This new item contains

- the collection tag specified by the `tag` parameter
- the collection ID specified by the `id` parameter
- the attributes specified by the default attributes of the `c` collection
- the variable-length data specified by the `itemSize` and `itemData` parameters

This function copies the information pointed to by the `itemData` parameter into the new item; after calling this function, you may alter this information or free the memory pointed to by this parameter without affecting the collection.

If the `c` collection already contains an item with the same collection tag and collection ID as specified in the `tag` and `id` parameters, this function removes the original item and replaces it with the new one, unless the existing item is locked. If it is locked, this function returns a `collectionItemLockedErr` result code.

The `itemSize` parameter determines how many bytes of information this function copies into the new item. If you specify 0 for this parameter, or provide `NULL` for the `itemData` parameter, this function copies no information into the variable-length data of the new item, or removes the variable-length data if the item already exists.

To lock a collection item, use the functions `SetCollectionItemInfo` (page 299) and `SetIndexedCollectionItemInfo` (page 300).

To replace a collection item using the item's index (rather than the item's tag and ID), use the `ReplaceIndexedCollectionItem` (page 295) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

AddCollectionItemHdl

Adds a new item to a collection or to replace an existing item in a collection, specifying the item's variable-length data using a handle rather than a pointer and a data size.

```
OSErr AddCollectionItemHdl (
   Collection aCollection,
   CollectionTag tag,
   SInt32 id,
   Handle itemData
);
```

Parameters

aCollection

A reference to the collection you want to add the item to. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag you want to associate with the new item.

id

The collection ID you want to associate with the new item.

itemData

A Macintosh Memory Manager handle to the item's variable-length data. This function copies the information referenced by the `itemData` parameter into the new item; after calling this function, you may alter this information or free the memory referenced by this parameter without affecting the collection.

Return Value

A result code. See "Result Codes" (page 313). If the `aCollection` collection already contains an item with the same collection tag and collection ID as specified in the `tag` and `id` parameters, this function removes the variable-length data from the original item and replaces it with the new data, unless the existing item is locked. If it is locked, this function returns a `collectionItemLockedErr` result code.

Discussion

The `AddCollectionItemHdl` function adds an item to the collection referenced by the `aCollection` parameter. This new item contains:

- the collection tag specified by the `tag` parameter
- the collection ID specified by the `id` parameter
- the attributes specified by the default attributes of the aCollection collection
- the variable-length data specified by the `itemData` parameter

To add or replace a collection item using a pointer (rather than a handle) to the item's variable-length data, use the `AddCollectionItem` (page 267) function.

To replace a collection item using the item's collection index (rather than the item's collection tag and collection ID), use the `ReplaceIndexedCollectionItemHdl` (page 297) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

CloneCollection

Clones a collection object—that is, increment its owner count.

```
Collection CloneCollection (
   Collection c
);
```

Parameters

c

A reference to the collection object you want to clone. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

Return Value

A reference to the cloned collection. (This result is effectively a copy of the reference you provide in the `c` parameter. See the description of the `Collection` data type.

Discussion

Typically, you use this function to increment a collection object's owner count to represent a new reference to the collection object. For example, if you want two variables in your application to reference a single collection object, you can use this code to maintain the correct owner count:

```
firstReference = NewCollection();
secondReference = CloneCollection(firstReference);
```

Disposing of either reference (using the `DisposeCollection` function) simply decrements the collection's owner count. Disposing of the remaining reference decrements the owner count again and frees the memory associated with the collection.

To decrement the owner count of a collection object, use the `DisposeCollection` (page 274) function. To determine the owner count of an existing collection object, use the `CountCollectionOwners` (page 272) function.

To copy a collection object, use the `CopyCollection` (page 271) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

CollectionTagExists

Determines whether any of the items in a specified collection contain a specified collection tag.

```
Boolean CollectionTagExists (
   Collection c,
   CollectionTag tag
);
```

Parameters

c

A reference to the collection object you want to search for a specific collection tag. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag to search for in the collection.

Return Value
`True` if the `c` collection contains any items that contain the specified tag.

Discussion
For information about data types related to collection tags, see `CollectionTag` (page 306).

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

CopyCollection

Creates a copy of an existing collection.

```
Collection CopyCollection (
   Collection srcCollection,
   Collection dstCollection
);
```

Parameters

srcCollection

A reference to the collection object you want to copy. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

dstCollection

A reference to a collection object to contain the copied collection items. You may provide `NULL` for this parameter to request that the Collection Manager create a new collection object to hold the copied information.

Return Value
A reference to the collection object containing the copied information. See the description of the `Collection` data type.

Discussion
The `CopyCollection` function copies all of the information (except the owner count and exception procedure) from the collection object referenced by the `srcCollection` parameter into the collection object referenced by the `dstCollection` parameter.

If you specify `NULL` for the `dstCollection` parameter, this function creates a new collection object to copy the information into. (This function does not return an error code; it returns `NULL` if it cannot create a new collection object.)

To clone a collection object, use the `DisposeCollection` (page 274) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

CountCollectionItems

Determines the total number of items in a collection.

```
SInt32 CountCollectionItems (
   Collection c
);
```

Parameters

c

A reference to the collection object whose items you want to count. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

Return Value

The total number of items in the `c` collection.

Discussion

To count the items in a collection that have a specified collection tag, use the `CountTaggedCollectionItems` (page 273) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

CountCollectionOwners

Determines the number of existing references to a collection object.

```
SInt32 CountCollectionOwners (
   Collection c
);
```

Parameters

c

The collection object whose owner count you want to determine. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

Return Value

The owner count of the collection object.

Discussion

To increment the owner count of a collection object, use the `CloneCollection` (page 270) function. To decrement the owner count of a collection object, use the `DisposeCollection` (page 274) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

CountCollectionTags

Determines the number of distinct collection tags contained by the items of a specified collection.

```
SInt32 CountCollectionTags (
   Collection c
);
```

Parameters

c

A reference to the collection object whose collection tags you want to count. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

Return Value

The number of distinct collection tags contained by the items of the `c` collection.

Discussion

For information about data types related to collection tags, see `CollectionTag` (page 306).

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

CountTaggedCollectionItems

Obtains the total number of items in a collection that have a specified collection tag.

```
SInt32 CountTaggedCollectionItems (
   Collection c,
   CollectionTag tag
);
```

Parameters

c

A reference to the collection object whose items you want to count. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag associated with the items you want to count.

Return Value

The total number of items in the `c` collection whose collection tags match the value specified in the `tag` parameter.

Discussion

To count all of the items in a collection, use the `CountCollectionItems` (page 272) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

DisposeCollection

Disposes of a collection object.

```
void DisposeCollection (
   Collection c
);
```

Parameters

c

A reference to the collection object you want to dispose of. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

Discussion

The `DisposeCollection` function decrements the owner count of the collection object referenced by the `c` parameter. If the resulting owner count is 0, this function releases the memory occupied by the collection object, and the collection object reference contained in the `c` parameter becomes invalid.

To create a new collection object, use the `NewCollection` (page 291) function.

To increment the owner count of a collection object, use the `CloneCollection` (page 270) function. To determine the owner count of an existing collection object, use the `CountCollectionOwners` (page 272) function

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

DisposeCollectionExceptionUPP

Disposes of a universal procedure pointer (UPP) to an error-handling callback.

```
void DisposeCollectionExceptionUPP (
   CollectionExceptionUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion

See the callback `CollectionExceptionProcPtr` (page 303) for more information.

Availability

Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

DisposeCollectionFlattenUPP

Disposes of a universal procedure pointer (UPP) to a data-flattening callback.

```
void DisposeCollectionFlattenUPP (
   CollectionFlattenUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion

See the callback `CollectionFlattenProcPtr` (page 304) for more information.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

EmptyCollection

Removes every item in a collection.

```
void EmptyCollection (
   Collection c
);
```

Parameters

c

A reference to the collection object you want to empty. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

Discussion

This function removes every item in the collection referenced by the `c` parameter. This function provides the fastest mechanism for emptying a collection.

To remove all of the items in a collection whose attributes match a specified pattern, use the `PurgeCollection` (page 292) function.

To remove all of the items in a collection with a specified collection tag, use the `PurgeCollectionTag` (page 293) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

FlattenCollection

Converts a collection object into a stream format suitable for storing and unflattening.

```
OSErr FlattenCollection (
   Collection c,
   CollectionFlattenUPP flattenProc,
   void *refCon
);
```

Parameters

c

A reference to the collection that you want to flatten. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

flattenProc

A pointer to a callback function you provide to process the flattened stream of bytes.

refCon

A pointer to the reference constant that you want the Collection Manager to pass to your callback function each time that it calls the callback function. You can use this parameter as a pointer to a structure containing information your callback function needs to process the blocks of flattened data.

Return Value
A result code. See "Result Codes" (page 313). This function can return any error returned by the callback function.

Discussion
You could, for example, use this function to copy a collection onto the Clipboard so that it could be pasted into another application.

The `FlattenCollection` function flattens into a stream of bytes the collection you specify with the `c` parameter. As this function flattens the collection, it repeatedly calls the callback function you specify using the `flattenProc` parameter. Each time it calls this function, it provides the callback function with a pointer to a block of memory containing flattened data. It continues to call this function until it has flattened the entire collection. Your callback function can process the flattened data in a number of ways: it could copy the flattened data into a handle-based block of memory, it could write the flattened data to disk, and so on.

When flattening the `c` collection, this function includes only the collection items whose persistence attribute is set.

To create a flattened collection that includes only those collection items whose attributes match a specified pattern, use the `FlattenPartialCollection` (page 277) function.

To unflatten a flattened collection, use the `UnflattenCollection` (page 301) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

FlattenCollectionToHdl

Flattens a collection into a Macintosh Memory Manager handle.

```
OSErr FlattenCollectionToHdl (
   Collection aCollection,
   Handle flattened
);
```

Parameters

aCollection

The collection that you want to flatten into a handle. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

flattened

A handle to contain the flattened data. You must provide a valid Macintosh Memory Manager handle in this parameter. You may specify a handle of size 0; this function resizes the handle as necessary to hold the flattened data.

Return Value
A result code. See "Result Codes" (page 313).

Discussion
This function flattens the collection referenced by the `aCollection` parameter into a block of memory referenced by the handle you provide in the `flattened` parameter.

To flatten a collection directly to disk, use the `FlattenCollection` (page 275) function.

To unflatten a collection from a block of memory referenced by a handle, use the `UnflattenCollectionFromHdl` (page 302) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

FlattenPartialCollection

Converts a collection object into a stream format suitable for storage and unflattening.

```
OSErr FlattenPartialCollection (
   Collection c,
   CollectionFlattenUPP flattenProc,
   void *refCon,
   SInt32 whichAttributes,
   SInt32 matchingAttributes
);
```

Parameters

c

The collection that you want to flatten. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

flattenProc

A pointer to a function to write data.

refCon
A reference constant that you want the Collection Manager to pass to your flatten function each time it calls the flatten function. You can use this parameter as a pointer to a structure containing information your callback function needs to process the blocks of flattened data.

whichAttributes
A mask indicating which attributes you want to test.

matchingAttributes
An `SInt32` word containing the attribute values you want to match.

Return Value
A result code. See "Result Codes" (page 313). This function can return any error returned by the callback function.

Discussion
With this function, you can include in the flattened collection only those items whose attributes match a specified pattern.

The `FlattenPartialCollection` function flattens into a stream of bytes the collection you specify with the `c` parameter. It includes only the collection items whose attributes specified by the `whichAttributes` parameter match the values specified by the `matchingAttributes` parameter.

As this function flattens the collection, it repeatedly calls the callback function you specify using the `flattenProc` parameter. Each time it calls this function, it provides the callback function with a pointer to a block of memory containing flattened data. It continues to call this function until it has flattened the entire collection. Your callback function can process the flattened data in a number of ways: it could copy the flattened data into a handle-based block of memory, it could write the flattened data to disk, and so on.

When flattening the c collection, this function includes only the collection items whose persistence attribute is set, regardless of the values you provide in the `whichAttributes` and `matchingAttributes` parameters.

To create a flattened collection that includes every item in a collection, use the `FlattenCollection` (page 275) function.

To unflatten a flattened collection, use the `UnflattenCollection` (page 301) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetCollectionDefaultAttributes

Examines the default attributes of a collection object.

```
SInt32 GetCollectionDefaultAttributes (
   Collection c
);
```

Parameters

c

A reference to the collection object whose default attributes you want to determine. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

Return Value
An `SInt32` word containing the bit flags that make up the collection's default attributes.

Discussion
To change the attributes of a collection object, use the `SetCollectionDefaultAttributes` (page 298) function.

To examine the attributes of a specific item in a collection, use `GetCollectionItemInfo` (page 281), `GetIndexedCollectionItemInfo` (page 285), and `GetTaggedCollectionItemInfo` (page 288)

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetCollectionExceptionProc

Obtains a pointer to the exception procedure installed in a specified collection.

```
CollectionExceptionUPP GetCollectionExceptionProc (
   Collection c
);
```

Parameters

c

A reference to the collection object whose exception procedure you want to determine. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

Return Value
A pointer to the exception procedure installed in the `c` collection object. See the description of the `CollectionExceptionUPP` data type.

Discussion
To install a new exception procedure in a collection object, use the `SetCollectionExceptionProc` (page 299) function.

For more information about exception procedures, see `CollectionExceptionProcPtr` (page 303).

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetCollectionItem

Obtains a copy of the variable-length data associated with a collection item given the item's collection tag and collection ID.

```
OSErr GetCollectionItem (
   Collection c,
   CollectionTag tag,
   SInt32 id,
   SInt32 *itemSize,
   void *itemData
);
```

Parameters

c

A reference to the collection object containing the item whose data you want to retrieve. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag associated with the item whose data you want to retrieve.

id

The collection ID associated with the item whose data you want to retrieve.

itemSize

A pointer to an `SInt32` value indicating the number of bytes of data you want returned in the `itemData` parameter. On return, this value indicates the size in bytes of the variable-length data associated with the specified item. You may specify the constant `dontWantSize` for this parameter to indicate that you want to copy all the specified item's variable-length data and you do not want to determine the size of this data. You may specify a value for the `itemSize` parameter that is greater than the actual number of bytes in the specified item's variable-length data however, this function never returns in the `itemData` parameter more data than contained in the specified item's variable-length data.

itemData

A pointer to a block of memory to contain the item's data. On return, this memory contains a copy of the data associated with the specified item. You may specify the constant `dontWantData` for this parameter if you do not want a copy of the item's data.

Return Value

A result code. See "Result Codes" (page 313).

Discussion

If you do not know the size of the item you want to retrieve, you typically call this function twice. The first time you provide a pointer in the `itemSize` parameter to determine the size of the specified item's data and you specify `dontWantData` for the `itemData` parameter. Then you allocate a memory block large enough to hold a copy of the item's data. Then you call the function a second time. This time you specify the constant `dontWantSize` for the `itemSize` parameter and provide a pointer to the allocated memory block for the `itemData` parameter. The function then copies the data into the allocated block of memory.

To retrieve the data associated with a collection item given its collection index (rather than its collection tag and ID), use the `GetIndexedCollectionItem` (page 283) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetCollectionItemHdl

Obtains a copy of the variable-length data associated with a collection item given the item's collection tag and collection ID.

```
OSErr GetCollectionItemHdl (
   Collection aCollection,
   CollectionTag tag,
   SInt32 id,
   Handle itemData
);
```

Parameters

aCollection
A reference to the collection object containing the item whose data you want to retrieve. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag
The collection tag associated with the item whose data you want to retrieve.

id
The collection ID associated with the item whose data you want to retrieve.

itemData
A handle to a block of memory to contain the item's data. On return, this memory contains a copy of the data associated with the specified item. You must provide a valid Macintosh Memory Manager handle for this function to copy the data into.You may specify the constant `dontWantData` for this parameter if you do not want a copy of the item's data.

Return Value
A result code. See "Result Codes" (page 313).

Discussion
You specify a collection object using the `aCollection` parameter and you specify an item in that collection using the `tag` and `id` parameters.

To retrieve the data associated with a collection item into a block of memory referenced by a pointer (rather than a handle), use the `GetCollectionItem` (page 280) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetCollectionItemInfo

Obtains information about a specific collection item given the item's collection tag and collection ID.

```
OSErr GetCollectionItemInfo (
   Collection c,
   CollectionTag tag,
   SInt32 id,
   SInt32 *index,
   SInt32 *itemSize,
   SInt32 *attributes
);
```

Parameters

c

A reference to the collection object containing the item you want to obtain information about. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag associated with the item you want to obtain information about.

id

The collection ID associated with the item you want to obtain information about.

index

On return, this value represents the collection index of the specified item. You may specify the constant `dontWantIndex` for this parameter if you do not want to determine the specified item's collection index.

itemSize

On return, this value indicates the size in bytes of the variable-length data associated with the specified item. You may specify the constant `dontWantSize` for this parameter to indicate that you do not want to determine the size of this data.

attributes

On return, this value contains a copy of the attributes associated with the specified item. You may specify the constant `dontWantAttributes` for this parameter if you do not want a copy of the item's attributes.

Return Value
A result code. See "Result Codes" (page 313).

Discussion
This function returns information in the `index`, `itemSize`, and `attributes` parameters:

- If you provide a pointer in the `index` parameter, the function uses this parameter to return the collection index of the specified item. Once you have determined an item's collection index, you can use it to specify the item when calling Collection Manager functions, rather than using the item's collection tag and collection ID. Specifying collection items using their collection index, rather than using the item's collection tag and collection ID, generally results in improved performance.
- If you provide a pointer in the `itemSize` parameter, the function uses this parameter to return the size in bytes of the variable-length data associated with the specified collection item.
- If you provide a pointer in the `attributes` parameter, the function uses this parameter to return a copy of the attributes associated with the specified collection item.

To obtain information about a collection item using the collection index to specify the item, use the `GetIndexedCollectionItemInfo` (page 285) function.

To obtain information about a collection item using the tag and `whichItem` parameters to specify the item, use the `GetTaggedCollectionItemInfo` (page 288) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetCollectionRetainCount

Obtains the owner count (the number of existing references) for a collection object.

```
ItemCount GetCollectionRetainCount (
   Collection c
);
```

Parameters

c

Discussion
This function performs the same operation as `CountCollectionOwners` (page 272), but follows the preferred naming conventions for Carbon and Core Foundation functions.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.1 and later.

Declared In
`Collections.h`

GetIndexedCollectionItem

Obtains a copy of the variable-length data associated with a collection item given the item's collection index.

```
OSErr GetIndexedCollectionItem (
   Collection c,
   SInt32 index,
   SInt32 *itemSize,
   void *itemData
);
```

Parameters

c

A reference to the collection object containing the item whose data you want to retrieve. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

index

The collection index associated with the item whose data you want to retrieve.

itemSize

A pointer to an `SInt32` value indicating the number of bytes of data you want returned in the `itemData` parameter. On return, this value indicates the size in bytes of the variable-length data associated with the specified item. You may specify the constant `dontWantSize` for this parameter to indicate that you want to copy all of the specified item's variable-length data and you do not want to determine the size of this data. You may specify a value for the `itemSize` parameter that is greater than the actual number of bytes in the specified item's variable-length data however, this function never returns in the `itemData` parameter more data than contained in the specified item's variable-length data.

itemData

A pointer to a block of memory to contain the item's data. On return, this memory contains a copy of the data associated with the specified item. You may specify the constant `dontWantData` for this parameter if you do not want a copy of the item's data.

Return Value
A result code. See "Result Codes" (page 313).

Discussion
If you do not know the size of the item you want to retrieve, you typically call this function twice. The first time you provide a pointer in the `itemSize` parameter to determine the size of the specified item's data and you specify the constant `dontWantData` for the `itemData` parameter. Then you allocate a memory block large enough to hold a copy of the item's data. Then you call the function a second time. This time you specify the constant `dontWantSize` for the `itemSize` parameter and provide a pointer to the allocated memory block for the `itemData` parameter. The function then copies the data into the allocated block of memory.

To retrieve the data associated with a collection item given its collection tag and ID (rather than its collection index), use the `GetCollectionItem` (page 280) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetIndexedCollectionItemHdl

Copies the variable-length data associated with a collection item into a Macintosh Memory Manager handle, given the item's collection index.

```
OSErr GetIndexedCollectionItemHdl (
   Collection aCollection,
   SInt32 index,
   Handle itemData
);
```

Parameters

aCollection

A reference to the collection object containing the item whose data you want to retrieve. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

index

The collection index associated with the item whose data you want to retrieve.

itemData

A handle to a block of memory to contain the item's data. On return, this memory contains a copy of the data associated with the specified item.

Return Value
A result code. See "Result Codes" (page 313).

Discussion
To retrieve the data associated with a collection item into a block of memory referenced by a pointer (rather than a handle), use the `GetCollectionItem` (page 280) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetIndexedCollectionItemInfo

Obtains information about a specific collection item given the item's collection index.

```
OSErr GetIndexedCollectionItemInfo (
   Collection c,
   SInt32 index,
   CollectionTag *tag,
   SInt32 *id,
   SInt32 *itemSize,
   SInt32 *attributes
);
```

Parameters

c

A reference to the collection object containing the item you want to obtain information about. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

index

The collection index associated with the item you want to obtain information about.

tag

A pointer to a collection tag. On return, the collection tag associated with the specified item. You may specify the constant `dontWantTag` for this parameter if you do not want to determine the specified item's collection tag.

id

A pointer to an `SInt32` value. On return, the collection ID associated with the specified item. You may specify the constant `dontWantId` for this parameter if you do not want to determine the specified item's collection ID.

itemSize

A pointer to an `SInt32` value. On return, this value indicates the size in bytes of the data associated with the specified item. You may specify the constant `dontWantSize` for this parameter if you do not want to determine the specified item's data size.

attributes

A pointer to an `SInt32` value. On return, this value contains a copy of the attributes associated with the specified item. You may specify the constant `dontWantAttributes` for this parameter if you do not want a copy of the item's attributes.

Return Value
A result code. See "Result Codes" (page 313).

Discussion
To obtain information about a collection item using the collection tag and collection ID to specify the item, use the `GetCollectionItemInfo` (page 281) function.

To obtain information about a collection item using the collection tag and tag list position to specify the item, use the `GetTaggedCollectionItemInfo` (page 288) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetIndexedCollectionTag

Examines a specific collection tag contained in a collection.

```
OSErr GetIndexedCollectionTag (
   Collection c,
   SInt32 tagIndex,
   CollectionTag *tag
);
```

Parameters

c

The collection from which to obtain a specific collection tag. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tagIndex

The position of the desired collection tag in the `c` collection's list of distinct collection tags.

tag

A pointer to a collection tag. On return, the collection tag that lies at the specified position in the list of distinct collection tags contained in the `c` collection.

Return Value
A result code. See "Result Codes" (page 313).

Discussion
Each collection object contains a number of distinct collection tags. By sequentially incrementing the value of the `tagIndex` parameter from 1 to the result of the `CountCollectionTags` (page 273) function, you can use this function to determine every collection tag contained in a collection.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetNewCollection

Reads a collection in from a collection (`'cltn'`) resource.

```
Collection GetNewCollection (
   SInt16 collectionID
);
```

Parameters

collectionID

The resource ID associated with the collection resource from which you want to create the new collection object.

Return Value
A reference to the new collection object. If this function does not find a collection resource with the specified resource ID, it returns `NULL` as the function result. See the description of the `Collection` data type.

Discussion
This function searches the current resource file path for a collection (`'cltn'`) resource with the resource ID specified by the `collectionID` parameter. If it finds such a resource, this function creates a new collection object, initializes it with the information stored in the resource, and returns a reference to it as the function result.

You can use the `MemError` and `ResError` functions to check for other errors after calling this function.

For information about collection resources, see 'cltn'.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetTaggedCollectionItem

Obtains a copy of the variable-length data associated with a collection item given the item's collection tag and tag list position.

```
OSErr GetTaggedCollectionItem (
   Collection c,
   CollectionTag tag,
   SInt32 whichItem,
   SInt32 *itemSize,
   void *itemData
);
```

Parameters

c

A reference to the collection object containing the item whose data you want to retrieve. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag
The collection tag associated with the item whose data you want to retrieve.

whichItem
The tag list position associated with the specific item.

itemSize
A pointer to an `SInt32` value indicating the number of bytes of data you want returned in the `itemData` parameter. On return, this value indicates the size in bytes of the variable-length data associated with the specified item. You may specify the constant `dontWantSize` for this parameter to indicate that you want to copy all of the specified item's variable-length data and you do not want to determine the size of this data.

itemData
A pointer to a block of memory to contain the item's data. On return, this memory contains a copy of the data associated with the specified item. You may specify the constant `dontWantData` for this parameter if you do not want a copy of the item's data.

Return Value
A result code. See "Result Codes" (page 313).

Discussion
Remember that a tag list position is the sequential index that determines an item given a specific collection tag. For example:

- A `whichItem` value of 1 indicates the first item with the specified tag.
- A `whichItem` value of 2 indicates the second item with the specified tag.

By sequentially incrementing the whichItem parameter, you can use this function to step through all of the items in a collection without knowing their collection IDs.

If you do not know the size of the item you want to retrieve, you typically call this function twice. The first time you provide a pointer in the `itemSize` parameter to determine the size of the specified item's data and you specify the constant `dontWantData` for the `itemData` parameter. Then you allocate a memory block large enough to hold a copy of the item's data. Then you call the function a second time. This time you specify the constant `dontWantSize` for the `itemSize` parameter and provide a pointer to the allocated memory block for the `itemData` parameter. The function then copies the data into the allocated block of memory.

To retrieve the data associated with a collection item given its collection tag and ID, use the `GetCollectionItem` (page 280) function.

To retrieve the data associated with a collection item given its collection index, use the `GetIndexedCollectionItem` (page 283) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

GetTaggedCollectionItemInfo

Obtains information about a specific collection item given the item's collection tag and tag list position.

```
OSErr GetTaggedCollectionItemInfo (
   Collection c,
   CollectionTag tag,
   SInt32 whichItem,
   SInt32 *id,
   SInt32 *index,
   SInt32 *itemSize,
   SInt32 *attributes
);
```

Parameters

c

A reference to the collection object containing the item you want to obtain information about. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag associated with the item you want to obtain information about.

whichItem

The tag list position of the item you want to obtain information about.

id

A pointer to an `SInt32` value. On return, this value represents the collection ID associated with the specified item. You may specify the constant `dontWantId` for this parameter if you do not want to determine the specified item's collection ID.

index

A pointer to an `SInt32` value. On return, this value represents the collection index of the specified item. You may specify the constant `dontWantIndex` for this parameter if you do not want to determine the specified item's collection index.

itemSize

A pointer to an `SInt32` value. On return, this value indicates the size in bytes of the data associated with the specified item. You may specify the constant `dontWantSize` for this parameter if you do not want to determine the specified item's data size.

attributes

A pointer. On return, this value contains a copy of the attributes associated with the specified item. You may specify the constant `dontWantAttributes` for this parameter if you do not want a copy of the item's attributes.

Return Value

A result code. See "Result Codes" (page 313).

Discussion

Remember that a collection tag and a tag list position uniquely identify a collection item. The tag list position indicates where the collection item would lie in a list made up of all the collection items with the same collection tag. For example:

- A `whichItem` value of 1 indicates the first item with the specified tag.
- A `whichItem` value of 2 indicates the second item with the specified tag.

By sequentially incrementing the `whichItem` parameter, you can use this function to step through all of the items in a collection that share a collection tag without knowing their collection IDs.

To obtain information about a collection item using the collection tag and collection ID to specify the item, use the `GetCollectionItemInfo` (page 281) function.

To obtain information about a collection item using the collection index to specify the item, use the `GetIndexedCollectionItemInfo` (page 285) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

InvokeCollectionExceptionUPP

Calls an error-handling callback.

```
OSErr InvokeCollectionExceptionUPP (
   Collection c,
   OSErr status,
   CollectionExceptionUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeCollectionExceptionUPP`, as the system calls your error-handling callback function for you. See the callback `CollectionExceptionProcPtr` (page 303) for more information.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

InvokeCollectionFlattenUPP

Calls a data-flattening callback.

```
OSErr InvokeCollectionFlattenUPP (
   SInt32 size,
   void *data,
   void *refCon,
   CollectionFlattenUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeCollectionFlattenUPP`, as the system calls your data-flattening callback function for you. See the callback `CollectionFlattenProcPtr` (page 304) for more information.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

NewCollection

Creates a new, empty collection object.

```
Collection NewCollection (
   void
);
```

Return Value
A reference to the newly created collection object. The new collection contains no items and has an owner count of 1. The `NewCollection` function does not return an error code; it returns `NULL` if it cannot create a new collection object. See the description of the `Collection` data type.

Discussion
The `NewCollection` function allocates memory for a new collection object, initializes it, and returns a reference to it.

To create a copy of an existing collection object, use the `CopyCollection` (page 271) function.

Special Considerations

You are responsible for disposing of collection objects that you create with this function when you no longer need them. To dispose of a collection object, use the `DisposeCollection` (page 274) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

NewCollectionExceptionUPP

Creates a new universal procedure pointer (UPP) to an error-handling callback.

```
CollectionExceptionUPP NewCollectionExceptionUPP (
   CollectionExceptionProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your error-handling callback.

Return Value
On return, a UPP to the error-handling callback. See the description of the `CollectionExceptionUPP` data type.

Discussion
See the callback `CollectionExceptionProcPtr` (page 303) for more information.

Availability
Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

NewCollectionFlattenUPP

Creates a new universal procedure pointer (UPP) to a data-flattening callback.

```
CollectionFlattenUPP NewCollectionFlattenUPP (
   CollectionFlattenProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your data-flattening callback.

Return Value
On return, a UPP to the data-flattening callback. See the description of the `CollectionFlattenUPP` data type.

Discussion
See the callback `CollectionFlattenProcPtr` (page 304) for more information.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

PurgeCollection

Removes all items in a collection whose attributes match a specified pattern.

```
void PurgeCollection (
   Collection c,
   SInt32 whichAttributes,
   SInt32 matchingAttributes
);
```

Parameters

c
A reference to the collection object containing the items you want to remove. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

whichAttributes
A mask indicating which attributes you want to test. You should set the bits of the parameter that correspond to the attributes you want to test.

matchingAttributes
An `SInt32` word containing the values of the attributes you want to match.

Discussion
The `PurgeCollection` function removes from the c collection any items whose attributes match the criteria you specify in the `whichAttributes` and `matchingAttributes` parameters.

This function compares the specified attributes of each item in the c collection with the corresponding attributes in the `matchingAttributes` parameter. If the values of all the specified attributes match, the function removes the item. To avoid purging locked items, you should clear the lock attribute in the `whichAttributes` and `matchingAttributes` parameters.

To remove all of the items in a collection with a specified collection tag, use the `PurgeCollectionTag` (page 293) function.

To remove every item in a collection, use the `EmptyCollection` (page 275) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

PurgeCollectionTag

Removes all items with a specific collection tag from a collection.

```
void PurgeCollectionTag (
   Collection c,
   CollectionTag tag
);
```

Parameters

c

A reference to the collection object containing the items you want to remove. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag associated with the items to remove.

Discussion
The `PurgeCollectionTag` function removes from the `c` collection all items whose collection tag matches the value of the `tag` parameter. This function removes locked and unlocked items.

To remove all of the items in a collection whose attributes match a specified pattern, use the `PurgeCollection` (page 292) function.

To remove every item in a collection, use the `EmptyCollection` (page 275) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

ReleaseCollection

Decrements the owner count (the number of existing references) for a collection object.

```
OSStatus ReleaseCollection (
   Collection c
);
```

Parameters

c

Return Value

A result code. See "Result Codes" (page 313).

Discussion

This function performs the same operation as `DisposeCollection` (page 274), but follows the preferred naming conventions for Carbon and Core Foundation functions.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X 10.1 and later.

Declared In

`Collections.h`

RemoveCollectionItem

Removes an item from a collection given the item's associated collection tag and collection ID.

```
OSErr RemoveCollectionItem (
   Collection c,
   CollectionTag tag,
   SInt32 id
);
```

Parameters

c

A reference to the collection object from which you want to remove the item. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag associated with the item you want to remove.

id

The collection ID associated with the item you want to remove.

Return Value

A result code. See "Result Codes" (page 313). If the `c` collection does not contain an item whose collection tag and collection ID match the values in the `tag` and `id` parameters, this function returns a `collectionItemNotFoundErr` result code.

Discussion

The `RemoveCollectionItem` function removes the item specified by the `tag` and `id` parameters from the collection referenced by the `c` parameter. This function removes the specified item even if its lock attribute is set.

To remove a collection item using the item's index (rather than the item's tag and ID), use the `RemoveIndexedCollectionItem` (page 295) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

RemoveIndexedCollectionItem

Removes an item from a collection given the item's index.

```
OSErr RemoveIndexedCollectionItem (
   Collection c,
   SInt32 index
);
```

Parameters

c

A reference to the collection object from which you want to remove the item. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

index

The collection index of the item you want to remove.

Return Value
A result code. See "Result Codes" (page 313). If the `c` collection does not contain an item whose collection index matches the values in the `index` parameter, this function returns a collectionIndexRangeErr result code.

Discussion
The `RemoveIndexedCollectionItem` function removes the item specified by the `index` parameter from the collection referenced by the `c` parameter. This function removes the specified item even if its lock attribute is set.

To remove a collection item using the item's tag and ID (rather than the item's index), use the `RemoveCollectionItem` (page 294) function.

To replace an item in a collection, use the function `ReplaceIndexedCollectionItem` (page 295).

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

ReplaceIndexedCollectionItem

Replaces the variable-length data of an item in a collection given the item's index.

```
OSErr ReplaceIndexedCollectionItem (
   Collection c,
   SInt32 index,
   SInt32 itemSize,
   const void *itemData
);
```

Parameters

c

A reference to the collection containing the item you want to replace. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

index

The collection index associated with the item to replace.

itemSize

The item's size. The `itemSize` parameter determines how many bytes of information this function copies into the new item. If you specify 0 for this parameter, or provide `NULL` for the `itemData` parameter, this function copies no information into the variable-length data of the new item, or removes the variable-length data if the item already exists.

itemData

A pointer to the item's data. This function copies the information pointed to by the `itemData` parameter into the new item; after calling this function, you may alter this information or free the memory pointed to by this parameter without affecting the collection.

Return Value
A result code. See "Result Codes" (page 313).

Discussion
You specify which item to replace using the `index` parameter. If the c collection does not contain an item whose collection index matches the value of the `index` parameter, this function returns a `collectionIndexRangeErr` result code.

If the c collection does contain an item with the specified index, this function replaces that item with a new item (if the existing item is not locked—if it is, this function returns a `collectionItemLockedErr` result code). The new item contains

- the same collection tag as the original item
- the same collection ID as the original item
- the same attributes as the original item
- the variable-length data specified by the `itemSize` and `itemData` parameters

To lock a collection item, use the functions `SetCollectionItemInfo` (page 299) and `SetIndexedCollectionItemInfo` (page 300).

To replace a collection item using the item's tag and ID (rather than the item's index), use the `AddCollectionItem` (page 267) function.

To remove an item from a collection, use the functions `RemoveCollectionItem` (page 294), `RemoveIndexedCollectionItem` (page 295), `PurgeCollection` (page 292), `PurgeCollectionTag` (page 293), and `EmptyCollection` (page 275).

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

ReplaceIndexedCollectionItemHdl

Replaces the variable-length data of an item in a collection given the item's collection index, specifying the item's new variable-length data using a handle rather than a pointer and a data size.

```
OSErr ReplaceIndexedCollectionItemHdl (
   Collection aCollection,
   SInt32 index,
   Handle itemData
);
```

Parameters

aCollection

A reference to the collection containing the item you want to replace. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

index

The collection index associated with the item you want to replace.

itemData

A Macintosh Memory Manager handle to the new variable-length data. This function copies the information referenced by the `itemData` parameter into the collection item; after calling this function, you may alter this information or free the memory referenced by this parameter without affecting the collection.

Return Value
A result code. See "Result Codes" (page 313). If the `aCollection` collection does not contain an item whose collection index matches the value of the `index` parameter, this function returns a `collectionIndexRangeErr` result code.

Discussion
If the `aCollection` collection does contain an item with the specified index, this function replaces the data in that item with new data (if the existing item is not locked—if it is, this function returns a `collectionItemLockedErr` result code). The resulting item contains

- the same collection tag as the original item
- the same collection ID as the original item
- the same attributes as the original item
- the variable-length data specified by the `itemData` parameter

To replace a collection item using a pointer (rather than a handle) to the item's variable-length data, use the `ReplaceIndexedCollectionItem` (page 295) function.

To replace a collection item using the item's collection tag and collection ID (rather than the item's collection index), use the `AddCollectionItemHdl` (page 268) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

RetainCollection

Increments the owner count (the number of existing references) for a collection object.

```
OSStatus RetainCollection (
   Collection c
);
```

Parameters

c

Return Value
A result code. See "Result Codes" (page 313).

Discussion
This function performs the same operation as `CloneCollection` (page 270), but follows the preferred naming conventions for Carbon and Core Foundation functions.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.1 and later.

Declared In
`Collections.h`

SetCollectionDefaultAttributes

Alters the default attributes of a collection object.

```
void SetCollectionDefaultAttributes (
   Collection c,
   SInt32 whichAttributes,
   SInt32 newAttributes
);
```

Parameters

c

A reference to the collection object whose default attributes you want to alter. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

whichAttributes

A mask indicating which bit flags in the c collection's default attributes you want to alter. For every bit in the `whichAttributes` parameter, this function takes one of two actions:

- If the bit is set, this function copies the value of the corresponding bit from the `newAttributes` parameter into the corresponding bit of the default attributes of the c collection.
- If the bit is not set, the corresponding bit of the c collection's default attributes remains unchanged.

newAttributes
The new values for the bit flags.

Discussion
To examine the attributes of a collection object, use the `GetCollectionDefaultAttributes` (page 278) function.

To change the attributes of a specific item in a collection, use the functions `SetCollectionItemInfo` (page 299) and `SetIndexedCollectionItemInfo` (page 300).

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

SetCollectionExceptionProc

Installs an exception procedure in a collection object.

```
void SetCollectionExceptionProc (
   Collection c,
   CollectionExceptionUPP exceptionProc
);
```

Parameters

c
A reference to the collection object whose exception procedure you want to change. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

exceptionProc
A pointer to the new exception procedure.

Discussion
The `SetCollectionExceptionProc` function copies the function pointer from the `exceptionProc` parameter into the collection object referenced by the `c` parameter.

To obtain a pointer to an existing exception procedure in a collection object, use the `GetCollectionExceptionProc` (page 279) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

SetCollectionItemInfo

Edits the attributes of a specific collection item given the item's collection tag and collection ID.

```
OSErr SetCollectionItemInfo (
   Collection c,
   CollectionTag tag,
   SInt32 id,
   SInt32 whichAttributes,
   SInt32 newAttributes
);
```

Parameters

c

A reference to the collection object containing the item whose attributes you want to edit. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

tag

The collection tag associated with the item whose attributes you want to edit.

id

The collection ID associated with the item whose attributes you want to edit.

whichAttributes

A mask indicating which attributes you want to edit.

newAttributes

An `SInt32` word containing the new settings for the attributes.

Return Value
A result code. See "Result Codes" (page 313).

Discussion
This function copies bit values from the `newAttributes` parameter to the attributes associated with the specified item.

This function uses the `whichAttributes` parameter to determine which bits to copy. For every bit in the `whichAttributes` parameter, this function takes one of two actions:

- If the bit is set, this function copies the value of the corresponding bit from the `newAttributes` parameter into the corresponding bit of the attributes associated with the specified item.
- If the bit is not set, the corresponding bit of the specified item's attributes remains unchanged.

The `whichAttributes` parameter allows you to change the values of specific bits in the specified item's attributes without affecting the values of other bits.

To obtain information about a collection item using the collection index to specify the item, use the `SetIndexedCollectionItemInfo` (page 300) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

SetIndexedCollectionItemInfo

Edits the attributes of a specific collection item given the item's collection index.

```
OSErr SetIndexedCollectionItemInfo (
   Collection c,
   SInt32 index,
   SInt32 whichAttributes,
   SInt32 newAttributes
);
```

Parameters

c

A reference to the collection object containing the item whose attributes you want to edit. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

index

The collection index of the item whose attributes you want to edit.

whichAttributes

A mask indicating which attributes you want to edit.

newAttributes

An `SInt32` word containing the new settings for the attributes.

Return Value
A result code. See "Result Codes" (page 313).

Discussion
The `SetIndexedCollectionItemInfo` function copies bit values from the `newAttributes` parameter to the attributes associated with the specified item.

This function uses the `whichAttributes` parameter to determine which bits to copy. For every bit in the `whichAttributes` parameter, this function takes one of two actions:

- If the bit is set, this function copies the value of the corresponding bit from the `newAttributes` parameter into the corresponding bit of the attributes associated with the specified item.
- If the bit is not set, the corresponding bit of the specified item's attributes remains unchanged.

The `whichAttributes` parameter allows you to change the values of specific bits in the specified item's attributes without affecting the values of other bits.

To edit the attributes of collection item using the collection tag and collection ID (rather than the collection index) to specify the item, use the `SetCollectionItemInfo` (page 299) function.

To examine the attributes of a collection item, use the functions `GetCollectionItemInfo` (page 281), `GetIndexedCollectionItemInfo` (page 285), and `GetTaggedCollectionItemInfo` (page 288).

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

UnflattenCollection

Unflattens a collection that was flattened using the `FlattenCollection` or `FlattenPartialCollection` function.

```
OSErr UnflattenCollection (
   Collection c,
   CollectionFlattenUPP flattenProc,
   void *refCon
);
```

Parameters

c

A reference to the collection object you want to create from the flattened data. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

flattenProc

A pointer to a function to read in flattened data.

refCon

A reference constant that you want the Collection Manager to pass to your callback function each time it calls the callback function. You can use this parameter as a pointer to a structure containing information your callback function needs when reading the blocks of flattened data.

Return Value

A result code. See "Result Codes" (page 313). This function can return any error returned by the callback function.

Discussion

The `UnflattenCollection` function unflattens a stream of bytes into the collection object you specify with the `c` parameter.

As this function unflattens the collection, it repeatedly calls the callback function you specify using the `flattenProc` parameter. Each time it calls this function, it provides the callback function with a pointer to a block of memory and a requested size. The callback function is responsible for reading the next set of bytes from the flattened byte stream and copying the data into the block of memory.

The Collection Manager continues to call your callback function, requesting more of the flattened stream of bytes each time, until it has unflattened the entire collection. Your callback function can read the flattened data from any source you choose: it could read the flattened data from a handle-based block of memory, it could read the flattened data from disk, and so on.

To create a flattened collection that includes only those collection items whose attributes match a specified pattern, use the `FlattenPartialCollection` (page 277) function.

To create a flattened collection that includes every item in a collection, use the `FlattenCollection` (page 275) function.

Availability

Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`Collections.h`

UnflattenCollectionFromHdl

Unflattens a collection that was flattened using the `FlattenCollectionToHdl` utility function.

```
OSErr UnflattenCollectionFromHdl (
   Collection aCollection,
   Handle flattened
);
```

Parameters

aCollection

A reference to a collection object in which to store the unflattened information. The behavior of this function is undefined if you do not provide a reference to a valid collection object.

flattened

A handle to the data that was previously flattened. You must provide a valid Macintosh Memory Manager handle in this parameter.

Return Value
A result code. See "Result Codes" (page 313).

Discussion
To unflatten a collection directly from disk, use the `UnflattenCollection` (page 301) function.

To flatten a collection to a block of memory referenced by a handle, use the `FlattenCollectionToHdl` (page 276) function.

Availability
Available in CarbonLib 1.0 and later when Collections 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`Collections.h`

Callbacks

CollectionExceptionProcPtr

Defines a pointer to an error handling callback function that handles errors that occur when operating on a collection object.

```
typedef OSErr (*CollectionExceptionProcPtr) (  Collection c,  OSErr status );
```

If you name your function `MyCollectionExceptionProc`, you would declare it like this:

```
OSErr MyCollectionExceptionProc (
    Collection c,
    OSErr status
);
```

Parameters

c

A reference to the collection object for which the error occurred.

status

The result code associated with the error that occurred.

Return Value
A result code. See "Result Codes" (page 313).

Discussion
You create this function to install in a collection object using the `SetCollectionExceptionProc` (page 299) function. Subsequently, whenever the Collection Manager is operating on that collection object and an error occurs, the Collection Manager calls this function, sending it a reference to the collection for which the error occurred and the result code associated with the error. You can use this information to handle the error appropriately for your application.

You can use an exception procedure to respond to an error in a number of ways:

- You can change the error from one result code to another by returning as the function result the new result code.
- You can handle the error and return the `noErr` error code, which indicates that the Collection Manager should return control to the place in your application that generated the error, as if no error had occurred.
- You can use the ANSI C functions `setjmp` and `longjmp` to jump out of the exception procedure into code to handle the error.

To install an exception procedure in a collection object, use the `SetCollectionExceptionProc` (page 299) function.

To obtain a pointer to an existing exception procedure in a collection object, use the `GetCollectionExceptionProc` (page 279) function.

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

Declared In
`Collections.h`

CollectionFlattenProcPtr

Defines a pointer to a flattening callback function that reads or writes flattened collection data.

```
typedef OSErr (*CollectionFlattenProcPtr)
(
    SInt32 size,
    void * data,
    void * refCon
);
```

If you name your function `MyCollectionFlattenProc`, you would declare it like this:

```
OSErr MyCollectionFlattenProc (
    SInt32 size,
    void * data,
    void * refCon
);
```

Parameters

size

The size of the block of flattened data to read or write.Your function should read or copy the requested number of bytes of flattened data into the block of memory pointed to by the `data` parameter.

data

A pointer to the block of flattened data. When flattening, this pointer points to the data your callback function should write. When unflattening, your callback function should read flattened data into the memory pointed to by this parameter.

refCon

A value you provide to the `FlattenCollection` function or `UnflattenCollection` function that the Collection Manager passes on to your callback function. You can use this parameter as a pointer to a structure containing relevant state information you need when reading or writing the flattened data.

Return Value

A result code. See "Result Codes" (page 313). If the execution of this function results in any fatal error, you should return the error code back to the Collection Manager as the function result. If the function executes successfully, you should return the `noErr` error code as the function result.

Discussion

You create this function to pass to the `FlattenCollection` (page 275), `FlattenPartialCollection` (page 277), and `UnflattenCollection` (page 301) functions when flattening or unflattening a collection.

As the Collection Manager is flattening a collection, it repeatedly calls this callback function to process sequential blocks of flattened data. Each time it calls this function, it provides a pointer to the current block of flattened data in the `data` parameter and the size of the current block in the `size` parameter. You can process this data in a number of ways: appending it to a handle-based block of memory, writing it to disk, and so on.

When unflattening a collection, the Collection Manager repeatedly calls this function to obtain blocks of flattened data.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Collections.h`

Data Types

Collection

Defines defines a reference to an opaque type that your compiler can type-check.

```
typedef struct OpaqueCollection * Collection;
```

Discussion

The Collection Manager provides you with access to a collection object through a `Collection` reference. The `Collection` type defines a reference type that your compiler can type-check; it does not define a pointer to a publicly defined data structure. The contents of the collection object are private; you must use the Collection Manager functions to manipulate collection objects.

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

Declared In
`Collections.h`

CollectionExceptionUPP

Defines a universal procedure pointer to an error-handling callback.

```
typedef CollectionExceptionProcPtr CollectionExceptionUPP;
```

Discussion
For more information, see the description of the `CollectionExceptionProcPtr` (page 303) callback function.

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

Declared In
`Collections.h`

CollectionFlattenUPP

Defines a universal procedure pointer to a data-flattening callback.

```
typedef CollectionFlattenProcPtr CollectionFlattenUPP;
```

Discussion
For more information, see the description of the `CollectionFlattenProcPtr` (page 304)callback function.

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

Declared In
`Collections.h`

CollectionTag

Defines a data type for a collection tag.

```
typedef FourCharCode CollectionTag;
```

Discussion
Each item in a collection is uniquely identified by its collection tag and its collection ID. The collection tag is a four-character identifier, similar to the identifiers used for resources.

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

Declared In
`Collections.h`

Constants

Attribute Bit Masks

Used to test or set a particular collection item attribute.

```
enum {
    kCollectionUser0Mask = 1L << kCollectionUser0Bit,
    kCollectionUser1Mask = 1L << kCollectionUser1Bit,
    kCollectionUser2Mask = 1L << kCollectionUser2Bit,
    kCollectionUser3Mask = 1L << kCollectionUser3Bit,
    kCollectionUser4Mask = 1L << kCollectionUser4Bit,
    kCollectionUser5Mask = 1L << kCollectionUser5Bit,
    kCollectionUser6Mask = 1L << kCollectionUser6Bit,
    kCollectionUser7Mask = 1L << kCollectionUser7Bit,
    kCollectionUser8Mask = 1L << kCollectionUser8Bit,
    kCollectionUser9Mask = 1L << kCollectionUser9Bit,
    kCollectionUser10Mask = 1L << kCollectionUser10Bit,
    kCollectionUser11Mask = 1L << kCollectionUser11Bit,
    kCollectionUser12Mask = 1L << kCollectionUser12Bit,
    kCollectionUser13Mask = 1L << kCollectionUser13Bit,
    kCollectionUser14Mask = 1L << kCollectionUser14Bit,
    kCollectionUser15Mask = 1L << kCollectionUser15Bit,
    kCollectionReserved0Mask = 1L << kCollectionReserved0Bit,
    kCollectionReserved1Mask = 1L << kCollectionReserved1Bit,
    kCollectionReserved2Mask = 1L << kCollectionReserved2Bit,
    kCollectionReserved3Mask = 1L << kCollectionReserved3Bit,
    kCollectionReserved4Mask = 1L << kCollectionReserved4Bit,
    kCollectionReserved5Mask = 1L << kCollectionReserved5Bit,
    kCollectionReserved6Mask = 1L << kCollectionReserved6Bit,
    kCollectionReserved7Mask = 1L << kCollectionReserved7Bit,
    kCollectionReserved8Mask = 1L << kCollectionReserved8Bit,
    kCollectionReserved9Mask = 1L << kCollectionReserved9Bit,
    kCollectionReserved10Mask = 1L << kCollectionReserved10Bit,
    kCollectionReserved11Mask = 1L << kCollectionReserved11Bit,
    kCollectionReserved12Mask = 1L << kCollectionReserved12Bit,
    kCollectionReserved13Mask = 1L << kCollectionReserved13Bit,
    kCollectionPersistenceMask = 1L << kCollectionPersistenceBit,
    kCollectionLockMask = 1L << kCollectionLockBit
};
```

Discussion

Using the attribute bit numbers, the Collection Manager provides convenient attribute masks for each of the attributes. You can use these attribute masks when testing or setting a particular collection item attribute.

Attribute Bit Masks (Old)

Used to test or set a particular collection item attribute.

```
enum {
    collectionUser0Mask = kCollectionUser0Mask,
    collectionUser1Mask = kCollectionUser1Mask,
    collectionUser2Mask = kCollectionUser2Mask,
    collectionUser3Mask = kCollectionUser3Mask,
    collectionUser4Mask = kCollectionUser4Mask,
    collectionUser5Mask = kCollectionUser5Mask,
    collectionUser6Mask = kCollectionUser6Mask,
    collectionUser7Mask = kCollectionUser7Mask,
    collectionUser8Mask = kCollectionUser8Mask,
    collectionUser9Mask = kCollectionUser9Mask,
    collectionUser10Mask = kCollectionUser10Mask,
    collectionUser11Mask = kCollectionUser11Mask,
    collectionUser12Mask = kCollectionUser12Mask,
    collectionUser13Mask = kCollectionUser13Mask,
    collectionUser14Mask = kCollectionUser14Mask,
    collectionUser15Mask = kCollectionUser15Mask,
    collectionReserved0Mask = kCollectionReserved0Mask,
    collectionReserved1Mask = kCollectionReserved1Mask,
    collectionReserved2Mask = kCollectionReserved2Mask,
    collectionReserved3Mask = kCollectionReserved3Mask,
    collectionReserved4Mask = kCollectionReserved4Mask,
    collectionReserved5Mask = kCollectionReserved5Mask,
    collectionReserved6Mask = kCollectionReserved6Mask,
    collectionReserved7Mask = kCollectionReserved7Mask,
    collectionReserved8Mask = kCollectionReserved8Mask,
    collectionReserved9Mask = kCollectionReserved9Mask,
    collectionReserved10Mask = kCollectionReserved10Mask,
    collectionReserved11Mask = kCollectionReserved11Mask,
    collectionReserved12Mask = kCollectionReserved12Mask,
    collectionReserved13Mask = kCollectionReserved13Mask,
    collectionPersistenceMask = kCollectionPersistenceMask,
    collectionLockMask = kCollectionLockMask
};
```

Discussion
Using the attribute bit numbers, the Collection Manager provides convenient attribute masks for each of the attributes. You can use these attribute masks when testing or setting a particular collection item attribute.

Attribute Bit Numbers

Provides constant names for each of the bits in a collection item attributes.

```
enum {
    kCollectionUser0Bit = 0,
    kCollectionUser1Bit = 1,
    kCollectionUser2Bit = 2,
    kCollectionUser3Bit = 3,
    kCollectionUser4Bit = 4,
    kCollectionUser5Bit = 5,
    kCollectionUser6Bit = 6,
    kCollectionUser7Bit = 7,
    kCollectionUser8Bit = 8,
    kCollectionUser9Bit = 9,
    kCollectionUser10Bit = 10,
    kCollectionUser11Bit = 11,
    kCollectionUser12Bit = 12,
    kCollectionUser13Bit = 13,
    kCollectionUser14Bit = 14,
    kCollectionUser15Bit = 15,
    kCollectionReserved0Bit = 16,
    kCollectionReserved1Bit = 17,
    kCollectionReserved2Bit = 18,
    kCollectionReserved3Bit = 19,
    kCollectionReserved4Bit = 20,
    kCollectionReserved5Bit = 21,
    kCollectionReserved6Bit = 22,
    kCollectionReserved7Bit = 23,
    kCollectionReserved8Bit = 24,
    kCollectionReserved9Bit = 25,
    kCollectionReserved10Bit = 26,
    kCollectionReserved11Bit = 27,
    kCollectionReserved12Bit = 28,
    kCollectionReserved13Bit = 29,
    kCollectionPersistenceBit = 30,
    kCollectionLockBit = 31
};
```

Discussion
The Collection Manager provides the attribute bit numbers enumeration to provide constant names for each of the bits in a collection item's attributes.

The lower 16 bits of the attributes property of a collection item represent the user-defined attributes. You can use these attributes for any purpose suitable to your application.

The upper 16 bits are reserved for use by Apple Computer, Inc. Currently, the 2 high bits are defined: bit 30 represents the persistence attribute and bit 31 represents the lock attribute.

Attribute Bit Numbers (Old)

Provides constant names for each of the bits in a collection item attributes.

```
enum {
    collectionUser0Bit = kCollectionUser0Bit,
    collectionUser1Bit = kCollectionUser1Bit,
    collectionUser2Bit = kCollectionUser2Bit,
    collectionUser3Bit = kCollectionUser3Bit,
    collectionUser4Bit = kCollectionUser4Bit,
    collectionUser5Bit = kCollectionUser5Bit,
    collectionUser6Bit = kCollectionUser6Bit,
    collectionUser7Bit = kCollectionUser7Bit,
    collectionUser8Bit = kCollectionUser8Bit,
    collectionUser9Bit = kCollectionUser9Bit,
    collectionUser10Bit = kCollectionUser10Bit,
    collectionUser11Bit = kCollectionUser11Bit,
    collectionUser12Bit = kCollectionUser12Bit,
    collectionUser13Bit = kCollectionUser13Bit,
    collectionUser14Bit = kCollectionUser14Bit,
    collectionUser15Bit = kCollectionUser15Bit,
    collectionReserved0Bit = kCollectionReserved0Bit,
    collectionReserved1Bit = kCollectionReserved1Bit,
    collectionReserved2Bit = kCollectionReserved2Bit,
    collectionReserved3Bit = kCollectionReserved3Bit,
    collectionReserved4Bit = kCollectionReserved4Bit,
    collectionReserved5Bit = kCollectionReserved5Bit,
    collectionReserved6Bit = kCollectionReserved6Bit,
    collectionReserved7Bit = kCollectionReserved7Bit,
    collectionReserved8Bit = kCollectionReserved8Bit,
    collectionReserved9Bit = kCollectionReserved9Bit,
    collectionReserved10Bit = kCollectionReserved10Bit,
    collectionReserved11Bit = kCollectionReserved11Bit,
    collectionReserved12Bit = kCollectionReserved12Bit,
    collectionReserved13Bit = kCollectionReserved13Bit,
    collectionPersistenceBit = kCollectionPersistenceBit,
    collectionLockBit = kCollectionLockBit
};
```

Discussion
The Collection Manager provides the attribute bit numbers enumeration to provide constant names for each of the bits in a collection item's attributes.

The lower 16 bits of the attributes property of a collection item represent the user-defined attributes. You can use these attributes for any purpose suitable to your application.

The upper 16 bits are reserved for use by Apple Computer, Inc. Currently, the 2 high bits are defined: bit 30 represents the persistence attribute and bit 31 represents the lock attribute.

Attributes Masks

Used to specify attributes for any of the attribute-related Collection Manager functions.

```
enum {
    kCollectionNoAttributes = 0x00000000,
    kCollectionAllAttributes = 0xFFFFFFFF,
    kCollectionUserAttributes = 0x0000FFFF,
    kCollectionDefaultAttributes = 0x40000000
};
```

Constants

`kCollectionNoAttributes`

Specifies a mask in which all collection attributes are clear. You might use this constant when clearing all the attributes of an item or when testing whether all of an item's attributes are clear.

Available in Mac OS X v10.0 and later.

Declared in `Collections.h`.

`kCollectionAllAttributes`

Specifies a mask in which all collection attributes are set. You might use this constant as a mask to indicate that you want to edit or test every attribute of an item, or you might use it to set every attribute of an item.

Available in Mac OS X v10.0 and later.

Declared in `Collections.h`.

`kCollectionUserAttributes`

Specifies a mask in which the user attributes are set and the reserved attributes are clear. You might use this constant as a mask to indicate that you want to edit or test only the user attributes of an item, or you might use it to set every user attribute of an item.

Available in Mac OS X v10.0 and later.

Declared in `Collections.h`.

`kCollectionDefaultAttributes`

Specifies a mask in which the persistent attribute is set and all other attributes are clear. You might use this constant when testing to see if an item's attributes have been edited.

Available in Mac OS X v10.0 and later.

Declared in `Collections.h`.

Discussion

The Collection Manager provides four convenient attributes masks that you can use when specifying attributes for any of the attribute-related Collection Manager functions. You can also use the attribute bit masks as masks for individual attributes.

Attributes Masks (Old)

Used to specify attributes for any of the attribute-related Collection Manager functions.

```
enum {
    noCollectionAttributes = kCollectionNoAttributes,
    allCollectionAttributes = kCollectionAllAttributes,
    userCollectionAttributes = kCollectionUserAttributes,
    defaultCollectionAttributes = kCollectionDefaultAttributes
};
```

Constants

`noCollectionAttributes`
Specifies a mask in which all collection attributes are clear. You might use this constant when clearing all the attributes of an item or when testing whether all of an item's attributes are clear.

Available in Mac OS X v10.0 and later.

Declared in `Collections.h`.

`allCollectionAttributes`
Specifies a mask in which all collection attributes are set. You might use this constant as a mask to indicate that you want to edit or test every attribute of an item, or you might use it to set every attribute of an item.

Available in Mac OS X v10.0 and later.

Declared in `Collections.h`.

`userCollectionAttributes`
Specifies a mask in which the user attributes are set and the reserved attributes are clear. You might use this constant as a mask to indicate that you want to edit or test only the user attributes of an item, or you might use it to set every user attribute of an item.

Available in Mac OS X v10.0 and later.

Declared in `Collections.h`.

`defaultCollectionAttributes`
Specifies a mask in which the persistent attribute is set and all other attributes are clear. You might use this constant when testing to see if an item's attributes have been edited.

Available in Mac OS X v10.0 and later.

Declared in `Collections.h`.

Discussion
The Collection Manager provides four convenient attributes masks that you can use when specifying attributes for any of the attribute-related Collection Manager functions. You can also use the attribute bit masks as masks for individual attributes.

Optional Return Value Constants

Used to specify that you do not want a particular piece of information.

```
enum {
    kCollectionDontWantTag = 0,
    kCollectionDontWantId = 0,
    kCollectionDontWantSize = 0,
    kCollectionDontWantAttributes = 0,
    kCollectionDontWantIndex = 0,
    kCollectionDontWantData = 0
};
```

Discussion
Many of the Collection Manager functions return multiple pieces of information. For most of these functions, you can specify that you do not want a specific piece of information to be returned by specifying `NULL` for the corresponding parameter when calling the function.

The Collection Manager provides the optional return value constants to make your code easier to read when specifying that you are not interested in obtaining certain types of information. You can use these enumeration constants in place of the more generic constant `NULL` when specifying that you do not want to receive certain optional return values from a function.

Optional Return Value Constants (Old)

Used to specify that you do not want a particular piece of information.

```
enum {
    dontWantTag = kCollectionDontWantTag,
    dontWantId = kCollectionDontWantId,
    dontWantSize = kCollectionDontWantSize,
    dontWantAttributes = kCollectionDontWantAttributes,
    dontWantIndex = kCollectionDontWantIndex,
    dontWantData = kCollectionDontWantData
};
```

Discussion
Many of the Collection Manager functions return multiple pieces of information. For most of these functions, you can specify that you do not want a specific piece of information to be returned by specifying `NULL` for the corresponding parameter when calling the function.

The Collection Manager provides the optional return value constants to make your code easier to read when specifying that you are not interested in obtaining certain types of information. You can use these enumeration constants in place of the more generic constant `NULL` when specifying that you do not want to receive certain optional return values from a function.

Result Codes

The most common result codes returned by the Collection Manager are listed in the table below.

Result Code	Value	Description
`collectionItemLockedErr`	-5750	Available in Mac OS X v10.0 and later.
`collectionItemNotFoundErr`	-5751	Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`collectionIndexRangeErr`	-5752	Available in Mac OS X v10.0 and later.
`collectionVersionErr`	-5753	Available in Mac OS X v10.0 and later.

Component Manager Reference

Framework: CoreServices/CoreServices.h

Declared in Components.h

Overview

You can use the Component Manager to allow your application to find and utilize various software objects (components) at run time. You can create your own components, and you can use the Component Manager to help manage your components. A component is a piece of code that provides a defined set of services to one or more clients. Applications, system extensions, as well as other components can use the services of a component. A component typically provides a specific type of service to its clients. For example, a component might provide image compression or image decompression capabilities; an application could call such a component, providing the image to compress, and the component could perform the desired operation and return the compressed image to the application. The Component Manager provides access to components and manages them by, for example, keeping track of the currently available components and routing requests to the appropriate component.

Functions by Task

Finding Components

`CountComponents` (page 330)
Returns the number of registered components that meet the selection criteria specified by your application.

`FindNextComponent` (page 333)
Returns the component identifier for the next registered component that meets the selection criteria specified by your application.

`GetComponentListModSeed` (page 338)
Allows your application to determine if the list of registered components has changed.

`GetComponentTypeModSeed` (page 341)

`ResolveComponentAlias` (page 353)

Opening and Closing Components

`OpenAComponent` (page 345)

`OpenADefaultComponent` (page 346)

`OpenComponent` (page 346)
Opens a connection to the component with the component identifier specified by your application.

`OpenDefaultComponent` (page 348)
Opens a connection to a registered component of the component type and subtype specified by your application.

`CloseComponent` (page 327)
Terminates your application's connection to a component.

Getting Information About Components

`GetComponentIconSuite` (page 334)
Returns a handle to a component's icon suite to your application.

`GetComponentInfo` (page 335)
Returns to your application the registration information for a component.

`GetComponentPublicIndString` (page 338)

`GetComponentPublicResource` (page 339)

`GetComponentPublicResourceList` (page 339)

`ComponentFunctionImplemented` (page 328) Deprecated in Mac OS X v10.5
Allows your application to determine whether a component supports a specified request.

`GetComponentVersion` (page 341) Deprecated in Mac OS X v10.5
Returns the version number of a component to your application.

Retrieving Component Errors

`GetComponentInstanceError` (page 336)
Returns to your application the last error generated by a specific connection to a component.

Calling Component Functions

`CallComponentOpen` (page 324)

`CallComponentClose` (page 320)

CallComponentCanDo (page 320)

CallComponentVersion (page 326)

CallComponentRegister (page 324)

CallComponentTarget (page 325)

CallComponentUnregister (page 325)

CallComponentDispatch (page 321)

CallComponentGetMPWorkFunction (page 323)

CallComponentGetPublicResource (page 324)

Accessing the Thread Safety Mode

CSSetComponentsThreadMode (page 331)
: Sets whether or not using thread-unsafe components is allowed in the current thread.

CSGetComponentsThreadMode (page 330)
: Indicates whether using thread-unsafe components is allowed in the current thread.

Creating and Managing Universal Procedure Pointers

NewComponentRoutineUPP (page 344)
: Creates a new universal procedure pointer (UPP) to a component routine callback function.

InvokeComponentRoutineUPP (page 342)
: Calls your component routine callback function

DisposeComponentRoutineUPP (page 333)
: Disposes of the universal procedure pointer (UPP) to a component routine callback function.

NewComponentFunctionUPP (page 343)

DisposeComponentFunctionUPP (page 332)

NewComponentMPWorkFunctionUPP (page 344)

InvokeComponentMPWorkFunctionUPP (page 342)

DisposeComponentMPWorkFunctionUPP (page 332)

NewGetMissingComponentResourceUPP (page 344)

InvokeGetMissingComponentResourceUPP (page 343)

DisposeGetMissingComponentResourceUPP (page 333)

Registering Components

RegisterComponent (page 348)
Registers a component stored in memory.

RegisterComponentResource (page 352)
Registers a component stored in a resource file.

RegisterComponentResourceFile (page 352)
Registers all component resources in the given resource file.

UnregisterComponent (page 357)
Removes a component from the Component Manager's registration list.

RegisterComponentFileRef (page 351)

RegisterComponentFileRefEntries (page 351)

RegisterComponentFile (page 350) Deprecated in Mac OS X v10.5

RegisterComponentFileEntries (page 350) Deprecated in Mac OS X v10.5

Dispatching to Component Functions

CallComponentFunction (page 321)
Invokes the specified function of your component.

CallComponentFunctionWithStorage (page 322)
Invokes the specified function of your component.

CallComponentFunctionWithStorageProcInfo (page 323)

Managing Component Connections

CountComponentInstances (page 329)
Determines the number of open connections being managed by a specified component.

GetComponentInstanceStorage (page 337)
Allows your component to retrieve a handle to the memory associated with a connection.

SetComponentInstanceStorage (page 354)
Allows your component to associate memory with a connection.

`ComponentSetTarget` (page 328) Deprecated in Mac OS X v10.5
Calls a component's target request function and informs a component that it has been targeted by another component.

Setting Component Errors

`SetComponentInstanceError` (page 353)
Passes error information to the Component Manager which sets the current error value for the appropriate connection.

Working With Component Reference Constants

`GetComponentRefcon` (page 339)
Retrieves the value of the reference constant for your component.

`SetComponentRefcon` (page 355)
Sets the reference constant for your component.

Accessing a Component's Resource File

`OpenAComponentResFile` (page 345)

`OpenComponentResFile` (page 347)
Allows your component to gain access to its resource file.

`CloseComponentResFile` (page 327)
Closes the resource file that your component opened previously with the `OpenComponentResFile` function.

`GetComponentResource` (page 340)

`GetComponentIndString` (page 335)

Calling Other Components

`DelegateComponentCall` (page 331)
Allows your component to pass on a request to a specified component.

Capturing Components

`CaptureComponent` (page 326)
Allows your component to capture another component.

`UncaptureComponent` (page 356)
Allows your component to uncapture a previously captured component.

Changing the Default Search Order

`SetDefaultComponent` (page 356)
Changes the search order for registered components.

Functions

CallComponentCanDo

```
ComponentResult CallComponentCanDo (
   ComponentInstance ci,
   SInt16 ftnNumber
);
```

Parameters

ci

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CallComponentClose

```
ComponentResult CallComponentClose (
   ComponentInstance ci,
   ComponentInstance self
);
```

Parameters

ci

self

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CallComponentDispatch

```
ComponentResult CallComponentDispatch (
   ComponentParameters *cp
);
```

Parameters

cp

Return Value

See the description of the `ComponentResult` data type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

CallComponentFunction

Invokes the specified function of your component.

```
ComponentResult CallComponentFunction (
   ComponentParameters *params,
   ComponentFunctionUPP func
);
```

Parameters

params

A pointer to the `ComponentDescription` (page 361) structure that your component received from the Component Manager. These are the parameters originally provided by the application that called your component.

func

A universal procedure pointer to the component function that is to handle the request. The Component Manager calls the function referred to by the `func` parameter, using Pascal calling conventions, with the parameters that were originally provided by the application that called your component. The function referred to by this parameter must return a function result of type `ComponentResult` indicating the success or failure of the operation. See the `ComponentRoutineProcPtr` (page 358) callback for more information on component functions.

Return Value

The value that is returned by the function referred to by the `func` parameter. Your component should use this value to set the current error for this connection. You can use the `SetComponentInstanceError` (page 353) function to set the current error.

Discussion

When an application requests service from your component, your component receives a component parameters structure containing the parameters that the application provided when it called your component. Your component can use this structure to access the parameters directly. Alternatively, you can use either this function or `CallComponentFunctionWithStorage` (page 322) to extract those parameters and pass them to a subroutine of your component. By taking advantage of these functions, you can simplify the structure of your component code.

If your component subroutine does not need global data, your component should use this function. If your component subroutine requires memory in which to store global data for the component, your component must use `CallComponentFunctionWithStorage`.

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

Declared In
`Components.h`

CallComponentFunctionWithStorage

Invokes the specified function of your component.

```
ComponentResult CallComponentFunctionWithStorage (
   Handle storage,
   ComponentParameters *params,
   ComponentFunctionUPP func
);
```

Parameters

storage

A handle to the memory associated with the current connection. The Component Manager provides this handle to your component along with the request.

params

A pointer to the `ComponentParameters` (page 365) structure that your component received from the Component Manager. These are the parameters originally provided by the application that called your component.

func

A universal procedure pointer to the component function that is to handle the request. The Component Manager calls the function referred to by the `func` parameter, using Pascal calling conventions, with the parameters that were originally provided by the application that called your component. These parameters are preceded by a handle to the memory associated with the current connection. The function referred to by the `func` parameter must return a function result of type `ComponentResult` indicating the success or failure of the operation. See the `ComponentRoutineProcPtr` (page 358) callback for more information on component functions.

Return Value
The value that is returned by the function referred to by the `func` parameter. Your component should use this value to set the current error for this connection. Use the `SetComponentInstanceError` (page 353) function to set the current error for a connection.

Discussion
When an application requests service from your component, your component receives a component parameters structure containing the parameters that the application provided when it called your component. Your component can use this structure to access the parameters directly. Alternatively, you can use either the `CallComponentFunction` (page 321) function or this function to extract those parameters and pass them to a subroutine of your component. By taking advantage of these functions, you can simplify the structure of your component code.

If your component subroutine requires a handle to the memory associated with the connection, you must use this function. You allocate the memory for a given connection each time your component is opened. You inform the Component Manager that a connection has memory associated with it by calling the `SetComponentInstanceError` (page 353) function.

Subroutines of a component that don't need global data should use `CallComponentFunction` instead.

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

Declared In
`Components.h`

CallComponentFunctionWithStorageProcInfo

```
ComponentResult CallComponentFunctionWithStorageProcInfo (
   Handle storage,
   ComponentParameters *params,
   ProcPtr func,
   ProcInfoType funcProcInfo
);
```

Parameters

storage

params

funcProcInfo

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

Declared In
`Components.h`

CallComponentGetMPWorkFunction

```
ComponentResult CallComponentGetMPWorkFunction (
   ComponentInstance ci,
   ComponentMPWorkFunctionUPP *workFunction,
   void **refCon
);
```

Parameters

ci

workFunction

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CallComponentGetPublicResource

```
ComponentResult CallComponentGetPublicResource (
   ComponentInstance ci,
   OSType resourceType,
   SInt16 resourceID,
   Handle *resource
);
```

Parameters

ci

resourceType

resource

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CallComponentOpen

```
ComponentResult CallComponentOpen (
   ComponentInstance ci,
   ComponentInstance self
);
```

Parameters

ci

self

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CallComponentRegister

```
ComponentResult CallComponentRegister (
   ComponentInstance ci
);
```

Parameters

ci

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CallComponentTarget

```
ComponentResult CallComponentTarget (
   ComponentInstance ci,
   ComponentInstance target
);
```

Parameters

ci

target

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CallComponentUnregister

```
ComponentResult CallComponentUnregister (
   ComponentInstance ci
);
```

Parameters

ci

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CallComponentVersion

```
ComponentResult CallComponentVersion (
   ComponentInstance ci
);
```

Parameters

ci

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

CaptureComponent

Allows your component to capture another component.

```
Component CaptureComponent (
   Component capturedComponent,
   Component capturingComponent
);
```

Parameters

capturedComponent

The component to be captured. Your component can obtain this identifier from the `FindNextComponent` (page 333) function or from the component registration functions. You can use a component instance here, but you must coerce the data type appropriately.

capturingComponent

Your component. Note that you can use the component instance (appropriately coerced) that your component received in its open request in this parameter.

Return Value
A new component identifier. Your component can use this new identifier to refer to the captured component. For example, your component can open the captured component by providing this identifier to the `OpenComponent` (page 346) structure. Your component must provide this identifier to the `UncaptureComponent` (page 356) function to specify the component to be restored to the search list. If the component you wish to capture is already captured, the component identifier is set to `NULL`. See the description of the `Component` data type.

Discussion
Typically, your component captures another component when you want to override all or some of the features provided by a component or to provide new features. For example, a component called NewMath might capture a component called OldMath. Suppose the NewMath component provides a new function, `DoExponent`. Whenever NewMath gets an exponent request, it can handle the request itself. For all other requests, NewMath might call the OldMath component to perform the request.

After capturing a component, your component might choose to target a particular instance of the captured component.

In response to this function, the Component Manager removes the specified component from the list of available components. As a result, applications cannot retrieve information about the captured component or gain access to it. Current clients of the captured component are not affected by this function.

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

Declared In
`Components.h`

CloseComponent

Terminates your application's connection to a component.

```
OSErr CloseComponent (
   ComponentInstance aComponentInstance
);
```

Parameters

aComponentInstance

The connection you wish to close. Your application obtains the component instance from the `OpenComponent` (page 346) function or the `OpenDefaultComponent` (page 348) function. You can use a component identifier here, but you must coerce the data type appropriately.

Return Value
A result code. See "Component Manager Result Codes" (page 380).

Discussion
This function closes only a single connection. If your application has several connections to a single component, you must call it once for each connection.

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

Related Sample Code
WhackedTV

Declared In
`Components.h`

CloseComponentResFile

Closes the resource file that your component opened previously with the `OpenComponentResFile` function.

```
OSErr CloseComponentResFile (
   ResFileRefNum refnum
);
```

Parameters

refnum

The reference number that identifies the resource file to be closed. Your component obtains this value from the `OpenComponentResFile` (page 347) function. Your component must close any open resource files before returning to the calling application.

Return Value
A result code. See "Component Manager Result Codes" (page 380).

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

Declared In
`Components.h`

ComponentFunctionImplemented

Allows your application to determine whether a component supports a specified request. (Deprecated in Mac OS X v10.5.)

```
ComponentResult ComponentFunctionImplemented (
   ComponentInstance ci,
   SInt16 ftnNumber
);
```

Parameters

ci

The component instance of which you wish to make a request. Your application obtains the component instance from the `OpenDefaultComponent` (page 348) function or the `OpenComponent` (page 346) function. You can use a component identifier here, but you must coerce the data type appropriately.

ftnNumber

A request code value. See the documentation supplied with the component for request code values.

Return Value
Indicates whether the component supports the specified request. You can interpret this number as if it were a Boolean value. If the returned value is `TRUE`, the component supports the specified request. If the returned value is `FALSE`, the component does not support the request. Your application can use this function to determine a component's capabilities.

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
`Components.h`

ComponentSetTarget

Calls a component's target request function and informs a component that it has been targeted by another component. (Deprecated in Mac OS X v10.5.)

```
ComponentResult ComponentSetTarget (
   ComponentInstance ci,
   ComponentInstance target
);
```

Parameters

ci

The component instance to which to send a target request (the component that has been targeted). You can use a component identifier here, but you must coerce the data type appropriately.

target

The component instance issuing the target request.

Return Value

The value that the targeted component instance returns in response to the target request, or `badComponentSelector` if the targeted component does not support the target request.

Discussion

Your component can target a component instance without capturing the component or your component can first capture the component and then target a specific instance of the component.

You should not target a component instance if the component does not support the target request. Before calling this function, you should issue a can do request to the component instance you want to target to verify that the component supports the target request. After receiving a target request, the targeted component instance should call the component instance that targeted it whenever the targeted component instance would normally call one of its defined functions.

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

`Components.h`

CountComponentInstances

Determines the number of open connections being managed by a specified component.

```
long CountComponentInstances (
   Component aComponent
);
```

Parameters

aComponent

The component for which you want a count of open connections. You can use a component instance here, but you must coerce the data type appropriately.

Return Value

The number of open connections for the specified component.

Discussion

This function can be useful if you want to restrict the number of connections for your component or if your component needs to perform special processing based on the number of open connections.

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

Related Sample Code
SoftVDigX

Declared In
`Components.h`

CountComponents

Returns the number of registered components that meet the selection criteria specified by your application.

```
long CountComponents (
   ComponentDescription *looking
);
```

Parameters

looking

A pointer to a `ComponentDescription` (page 361) structure. Your application specifies the criteria for the component search in the fields of this structure.

The Component Manager ignores fields in the component description structure that are set to 0. For example, if you set all the fields to 0, the Component Manager returns the number of components registered in the system. Similarly, if you set all fields to 0 except for the `componentManufacturer` field, the Component Manager returns the number of registered components supplied by the manufacturer you specify.

Return Value
The number of components that meet the specified search criteria.

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

Declared In
`Components.h`

CSGetComponentsThreadMode

Indicates whether using thread-unsafe components is allowed in the current thread.

```
CSComponentsThreadMode CSGetComponentsThreadMode (
   void
);
```

Return Value
A flag that indicates whether using thread-unsafe components is allowed in the current thread.

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

Declared In
`Components.h`

CSSetComponentsThreadMode

Sets whether or not using thread-unsafe components is allowed in the current thread.

```
void CSSetComponentsThreadMode (
   CSComponentsThreadMode mode
);
```

Parameters

mode

A flag that determines whether the current thread is restricted to calling components that are thread-safe. You should set this flag to `kCSAcceptThreadSafeComponentsOnlyMode` whenever you want the current thread to call only components that are thread-safe.

Discussion

Core Services maintains a component thread-mode flag for each thread in the current process. The default value of this flag is `kCSAcceptAllComponentsMode`, which means the thread can call any component regardless of whether the component is thread-safe. Applications and other high-level code that call component-based APIs (such as QuickTime) from preemptive threads should call this function from their thread beforehand and pass in the value `kCSAcceptThreadSafeComponentsOnlyMode`.

A thread's component thread-mode flag can safely retain its default value only if the thread is the main thread or if it participates in cooperative locking, such as Carbon Thread Manager-style cooperative threads and application threads that perform their own private locking.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`Components.h`

DelegateComponentCall

Allows your component to pass on a request to a specified component.

```
ComponentResult DelegateComponentCall (
   ComponentParameters *originalParams,
   ComponentInstance ci
);
```

Parameters

originalParams

A pointer to the `ComponentParameters` (page 365) structure provided to your component by the Component Manager.

ci

The component instance that is to process the request. The Component Manager provides a component instance to your component when it opens a connection to another component with the `OpenComponent` (page 346) or `OpenDefaultComponent` (page 348) function. You must specify a component instance; this function does not accept a component identifier.

Return Value

The component result returned by the specified component.

Discussion
Your component may supplement its capabilities by using the services of another component to directly satisfy application requests using this function. For example, you might want to create two similar components that provide different levels of service to applications. Rather than completely implementing both components, you could design one to rely on the capabilities of the other. In this manner, you have to implement only that portion of the more capable component that provides additional services.

You may also invoke the services of another component using the standard mechanisms used by applications. The Component Manager then passes the requests to the appropriate component, and your component receives the results of those requests.

Your component must open a connection to the component to which the requests are to be passed. Your component must close that connection when it has finished using the services of the other component.

Your component should never use this function with open or close requests from the Component Manager—always use the `OpenComponent` and `CloseComponent` (page 327) functions to manage connections with other components.

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

Declared In
`Components.h`

DisposeComponentFunctionUPP

```
void DisposeComponentFunctionUPP (
   ComponentFunctionUPP userUPP
);
```

Parameters
userUPP

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

Declared In
`Components.h`

DisposeComponentMPWorkFunctionUPP

```
void DisposeComponentMPWorkFunctionUPP (
   ComponentMPWorkFunctionUPP userUPP
);
```

Parameters
userUPP

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

Declared In
`Components.h`

DisposeComponentRoutineUPP

Disposes of the universal procedure pointer (UPP) to a component routine callback function.

```
void DisposeComponentRoutineUPP (
   ComponentRoutineUPP userUPP
);
```

Parameters

userUPP

Discussion

See the `ComponentRoutineProcPtr` (page 358) callback for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

DisposeGetMissingComponentResourceUPP

```
void DisposeGetMissingComponentResourceUPP (
   GetMissingComponentResourceUPP userUPP
);
```

Parameters

userUPP

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

FindNextComponent

Returns the component identifier for the next registered component that meets the selection criteria specified by your application.

```
Component FindNextComponent (
   Component aComponent,
   ComponentDescription *looking
);
```

Parameters

aComponent

The starting point for the search. Set this field to 0 to start the search at the beginning of the component list. If you are continuing a search, you can specify a component identifier previously returned by this function. The function then searches the remaining components.

looking

A pointer to a `ComponentDescription` (page 361) structure. Your application specifies the criteria for the component search in the fields of this structure.

The Component Manager ignores fields in the component description structure that are set to 0. For example, if you set all the fields to 0, all components meet the search criteria. In this case, your application can retrieve information about all of the components that are registered in the system by repeatedly calling `FindNextComponent` and `GetComponentInfo` (page 335) until the search is complete. Similarly, if you set all fields to 0 except for the `componentManufacturer` field, the Component Manager searches all registered components for a component supplied by the manufacturer you specify. Note that this function does not modify the contents of the component description structure you supply. To retrieve detailed information about a component, you need to use the `GetComponentInfo` (page 335) function to get the component description structure for each returned component.

Return Value

The component identifier of a component that meets the search criteria or 0 when there are no more matching components. Your application can use the component identifier returned by this function to get more information about the component, using `GetComponentInfo`, or to open the component, using either the `OpenDefaultComponent` (page 348) function or the `OpenComponent` (page 346) function. See the description of the `Component` data type.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

SoftVDigX

WhackedTV

Declared In

`Components.h`

GetComponentIconSuite

Returns a handle to a component's icon suite to your application.

```
OSErr GetComponentIconSuite (
   Component aComponent,
   Handle *iconSuite
);
```

Parameters

aComponent

The component whose icon suite you wish to obtain. Your application obtains a component identifier from the `FindNextComponent` (page 333) function. If your application registers a component, it can also obtain a component identifier from the `RegisterComponent` (page 348) or `RegisterComponentResource` (page 352) function. You can use a component instance here, but you must coerce the data type appropriately.

iconSuite

On return, a pointer to a handle for the component's icon suite or, if the component has not provided an icon suite, `NULL`. A component provides the resource ID of its icon family to the Component Manager in the optional extensions to the component resource. Your application is responsible for disposing of the returned icon suite handle.

Return Value
A result code. See "Component Manager Result Codes" (page 380).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Components.h`

GetComponentIndString

```
OSErr GetComponentIndString (
   Component aComponent,
   Str255 theString,
   SInt16 strListID,
   SInt16 index
);
```

Parameters

aComponent

theString

Return Value
A result code. See "Component Manager Result Codes" (page 380).

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

Related Sample Code
SoftVDigX

Declared In
`Components.h`

GetComponentInfo

Returns to your application the registration information for a component.

```
OSErr GetComponentInfo (
   Component aComponent,
   ComponentDescription *cd,
   Handle componentName,
   Handle componentInfo,
   Handle componentIcon
);
```

Parameters

aComponent

The component about which you wish to obtain information. Your application obtains a component identifier from the `FindNextComponent` (page 333) function. If your application registers a component, it can also obtain a component identifier from the `RegisterComponent` (page 348) or `RegisterComponentResource` (page 352) function.

You may supply a component instance rather than a component identifier to this function, but you must coerce the data type appropriately. Your application can obtain a component instance from the `OpenComponent` (page 346) or `OpenDefaultComponent` (page 348) functions.

cd

A pointer to a `ComponentDescription` (page 361) structure. The function returns information about the specified component in this structure.

componentName

On return, a handle to the component's name. If the component does not have a name, an empty handle. Set this field to `NULL` if you do not want to receive the component's name.

componentInfo

On return, a handle to the component's information string. If the component does not have an information string, an empty handle. Set this field to `NULL` if you do not want to receive the component's information string.

componentIcon

On return, a handle to the component's icon. If the component does not have an icon, an empty handle. Set this field to `NULL` if you do not want to receive the component's icon. To get a handle to the component's icon suite, if it provides one, use the `GetComponentIconSuite` (page 334) function.

Return Value
A result code. See "Component Manager Result Codes" (page 380).

Discussion
For information on registering components, see "Registering Components".

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

Related Sample Code
WhackedTV

Declared In
`Components.h`

GetComponentInstanceError

Returns to your application the last error generated by a specific connection to a component.

```
OSErr GetComponentInstanceError (
   ComponentInstance aComponentInstance
);
```

Parameters

aComponentInstance

The component instance from which you want error information. Your application obtains the component instance from the `OpenDefaultComponent` (page 348) function or the `OpenComponent` (page 346) function. You can use a component identifier here, but you must coerce the data type appropriately.

Return Value
A result code. See "Component Manager Result Codes" (page 380).

Discussion
Some component functions return error information as their function result. Other component functions set an error code that your application can retrieve using this function. Refer to the documentation supplied with the component for information on how that particular component handles errors.

Once you have retrieved an error code, the Component Manager clears the error code for the connection. If you want to retain that error value, you should save it in your application's local storage.

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

Declared In
`Components.h`

GetComponentInstanceStorage

Allows your component to retrieve a handle to the memory associated with a connection.

```
Handle GetComponentInstanceStorage (
   ComponentInstance aComponentInstance
);
```

Parameters

aComponentInstance

The connection for which to retrieve the associated memory. The Component Manager provides a component instance to your component when the connection is opened. You can use a component identifier here, but you must coerce the data type appropriately.

Return Value
A handle to the memory associated with the specified connection.

Discussion
Typically, your component does not need to use this function, because the Component Manager provides this handle to your component each time the client application requests service from this connection.

Your component tells the Component Manager about the memory associated with a connection by calling the `SetComponentInstanceStorage` (page 354) function.

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

Declared In
`Components.h`

GetComponentListModSeed

Allows your application to determine if the list of registered components has changed.

```
SInt32 GetComponentListModSeed (
   void
);
```

Parameters

Return Value
The component registration seed number. Each time the Component Manager registers or unregisters a component it generates a new, unique seed number. By comparing the return value to values previously returned by this function, you can determine whether the list has changed. Your application may use this information to rebuild its internal component lists or to trigger other activity that is necessary whenever new components are available.

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

Declared In
`Components.h`

GetComponentPublicIndString

```
OSErr GetComponentPublicIndString (
   Component aComponent,
   Str255 theString,
   SInt16 strListID,
   SInt16 index
);
```

Parameters
aComponent
theString

Return Value
A result code. See "Component Manager Result Codes" (page 380).

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

Declared In
`Components.h`

GetComponentPublicResource

```
OSErr GetComponentPublicResource (
   Component aComponent,
   OSType resourceType,
   SInt16 resourceID,
   Handle *theResource
);
```

Parameters

aComponent

resourceType

theResource

Return Value

A result code. See "Component Manager Result Codes" (page 380).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

GetComponentPublicResourceList

```
OSErr GetComponentPublicResourceList (
   OSType resourceType,
   SInt16 resourceID,
   SInt32 flags,
   ComponentDescription *cd,
   GetMissingComponentResourceUPP missingProc,
   void *refCon,
   void *atomContainerPtr
);
```

Parameters

resourceType

cd

missingProc

Return Value

A result code. See "Component Manager Result Codes" (page 380).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

GetComponentRefcon

Retrieves the value of the reference constant for your component.

```
long GetComponentRefcon (
   Component aComponent
);
```

Parameters

aComponent

The component whose reference constant you wish to get. You can use a component instance here, but you must coerce the data type appropriately.

Return Value
The reference constant for the specified component.

Discussion
There is one reference constant for each component, regardless of the number of connections to that component. When your component is registered, the Component Manager sets this reference constant to 0.

The reference constant is a 4-byte value that your component can use in any way you decide. For example, you might use the reference constant to store the address of a data structure that is shared by all connections maintained by your component. You should allocate shared structures in the system heap. Your component should deallocate the structure when its last connection is closed or when it is unregistered.

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

Declared In
Components.h

GetComponentResource

```
OSErr GetComponentResource (
   Component aComponent,
   OSType resType,
   SInt16 resID,
   Handle *theResource
);
```

Parameters

aComponent

resType

theResource

Return Value
A result code. See "Component Manager Result Codes" (page 380).

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

Declared In
Components.h

GetComponentTypeModSeed

```
SInt32 GetComponentTypeModSeed (
   OSType componentType
);
```

Parameters

componentType

Availability

Available in Mac OS X v10.0 and later.

Declared In

Components.h

GetComponentVersion

Returns the version number of a component to your application. (Deprecated in Mac OS X v10.5.)

```
ComponentResult GetComponentVersion (
   ComponentInstance ci
);
```

Parameters

ci

The component instance from which you want to retrieve version information. Your application obtains the component instance from the `OpenDefaultComponent` (page 348) function or the `OpenComponent` (page 346) function.

Return Value

The version number of the component you specify. The high-order 16 bits represent the major version, and the low-order 16 bits represent the minor version. The major version specifies the component specification level the minor version specifies a particular implementation's version number.

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

Components.h

InvokeComponentMPWorkFunctionUPP

```
ComponentResult InvokeComponentMPWorkFunctionUPP (
   void *globalRefCon,
   ComponentMPWorkFunctionHeaderRecordPtr header,
   ComponentMPWorkFunctionUPP userUPP
);
```

Parameters

header

userUPP

Return Value

See the description of the `ComponentResult` data type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

InvokeComponentRoutineUPP

Calls your component routine callback function

```
ComponentResult InvokeComponentRoutineUPP (
   ComponentParameters *cp,
   Handle componentStorage,
   ComponentRoutineUPP userUPP
);
```

Parameters

cp

componentStorage

userUPP

Return Value

See the description of the `ComponentResult` data type.

Discussion

See the `ComponentRoutineProcPtr` (page 358) callback for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

InvokeGetMissingComponentResourceUPP

```
OSErr InvokeGetMissingComponentResourceUPP (
   Component c,
   OSType resType,
   SInt16 resID,
   void *refCon,
   Handle *resource,
   GetMissingComponentResourceUPP userUPP
);
```

Parameters

c

resType

resource

userUPP

Return Value

A result code. See "Component Manager Result Codes" (page 380).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

NewComponentFunctionUPP

```
ComponentFunctionUPP NewComponentFunctionUPP (
   ProcPtr userRoutine,
   ProcInfoType procInfo
);
```

Parameters

procInfo

Return Value

See the description of the `ComponentFunctionUPP` data type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

NewComponentMPWorkFunctionUPP

```
ComponentMPWorkFunctionUPP NewComponentMPWorkFunctionUPP (
   ComponentMPWorkFunctionProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `ComponentMPWorkFunctionUPP` data type.

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

Declared In
`Components.h`

NewComponentRoutineUPP

Creates a new universal procedure pointer (UPP) to a component routine callback function.

```
ComponentRoutineUPP NewComponentRoutineUPP (
   ComponentRoutineProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `ComponentRoutineUPP` data type.

Discussion
See the `ComponentRoutineProcPtr` (page 358) callback for more information.

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

Declared In
`Components.h`

NewGetMissingComponentResourceUPP

```
GetMissingComponentResourceUPP NewGetMissingComponentResourceUPP (
   GetMissingComponentResourceProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `GetMissingComponentResourceUPP` data type.

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

Declared In
`Components.h`

OpenAComponent

```
OSErr OpenAComponent (
   Component aComponent,
   ComponentInstance *ci
);
```

Parameters

aComponent

ci

Return Value
A result code. See "Component Manager Result Codes" (page 380).

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

Declared In
`Components.h`

OpenAComponentResFile

```
OSErr OpenAComponentResFile (
   Component aComponent,
   ResFileRefNum *resRef
);
```

Parameters

aComponent

Return Value
A result code. See "Component Manager Result Codes" (page 380).

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

Declared In
`Components.h`

OpenADefaultComponent

```
OSErr OpenADefaultComponent (
   OSType componentType,
   OSType componentSubType,
   ComponentInstance *ci
);
```

Parameters

componentType

componentSubType

ci

Return Value

A result code. See "Component Manager Result Codes" (page 380).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

WhackedTV

Declared In

`Components.h`

OpenComponent

Opens a connection to the component with the component identifier specified by your application.

```
ComponentInstance OpenComponent (
   Component aComponent
);
```

Parameters

aComponent

The component you wish to open. Your application obtains this identifier from the `FindNextComponent` (page 333) function. If your application registers a component, it can also obtain a component identifier from the `RegisterComponent` function or the `RegisterComponentResource` function.

Return Value

A component instance which identifies your application's connection to the component. You must supply this component instance whenever you call the functions provided by the component. When you close the component, you must also supply this component instance to the `CloseComponent` (page 327) function.

If it cannot open the specified component, the function returns `NULL`.

See the description of the `ComponentInstance` data type.

Discussion

Your application must open a component before it can call any component functions. To use this function, you must already have obtained a component identifier. Alternatively, you can use the `OpenDefaultComponent` (page 348) function to open a component without calling `FindNextComponent`.

Note that your application may maintain several connections to a single component, or it may have connections to several components at the same time.

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

Related Sample Code
WhackedTV

Declared In
`Components.h`

OpenComponentResFile

Allows your component to gain access to its resource file.

```
ResFileRefNum OpenComponentResFile (
   Component aComponent
);
```

Parameters

aComponent

The component whose resource file you wish to open. Applications that register components may obtain this identifier from the `RegisterComponentResource` (page 352) function. You can use a component instance here, but you must coerce the data type appropriately.

Return Value
A reference number that your component can use to read data from the appropriate resource file. If the specified component does not have an associated resource file or if the Component Manager cannot open the resource file, the function returns 0 or a negative number.

Discussion
This function opens the resource file with read-only permission. The Component Manager adds the resource file to the current resource chain. Your component must close the resource file with the `CloseComponentResFile` (page 327) function before returning to the calling application. Note that there is only one resource file associated with a component.

Your component can use `FSpOpenResFile` or equivalent Resource Manager functions to open other resource files, but you must use this function to open your component's resource file.

If you store your component in a component resource but register the component with the `RegisterComponent` (page 348) function, rather than with the `RegisterComponentResource` or `RegisterComponentResourceFile` function, your component cannot access its resource file with this function.

Note that when working with resources, your component should always first save the current resource file, perform any resource operations, then restore the current resource file to its previous value before returning.

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

Declared In
`Components.h`

OpenDefaultComponent

Opens a connection to a registered component of the component type and subtype specified by your application.

```
ComponentInstance OpenDefaultComponent (
   OSType componentType,
   OSType componentSubType
);
```

Parameters

componentType

The type of the component. All components of a particular type support a common set of interface functions. Use this parameter to search for components of a given type.

componentSubType

The subtype of the component. Different subtypes of a component type may support additional features or provide interfaces that extend beyond the standard functions for a given component type. For example, the subtype of an image compressor component indicates the compression algorithm employed by the compressor.

Your application can use the `componentSubType` parameter to perform a more specific lookup operation than is possible using only the `componentType` parameter. For example, you may want your application to use only components of a certain component type (`'draw'`) that also have a specific subtype (`'oval'`). Set this parameter to 0 to select a component with any subtype value.

Return Value

A component instance that identifies the connection opened to the component which matches your search criteria. You must supply this component instance whenever you call the functions provided by the component. When you close the component, you must also supply this component instance to the `CloseComponent` (page 327) function.

If more than one component in the list of registered components meets the search criteria, the function opens the first one that it finds in its list. If it cannot open the specified component, it returns `NULL`.

See the description of the `ComponentInstance` data type.

Discussion

Your application must open a component before it can call any component functions. This function searches for a component by type and subtype. You do not have to supply a component description structure or call the `FindNextComponent` (page 333) function to use this function. If you want to exert more control over the selection process, you can use the `FindNextComponent` and `OpenComponent` (page 346) functions.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

RegisterComponent

Registers a component stored in memory.

```
Component RegisterComponent (
   ComponentDescription *cd,
   ComponentRoutineUPP componentEntryPoint,
   SInt16 global,
   Handle componentName,
   Handle componentInfo,
   Handle componentIcon
);
```

Parameters

cd

A pointer to a `ComponentDescription` (page 361) structure that describes the component to be registered. You must correctly fill in the fields of this structure before calling this function. When applications search for components using the `FindNextComponent` (page 333) function, the Component Manager compares the attributes you specify here with those specified by the application. If the attributes match, the Component Manager returns the component identifier to the application.

componentEntryPoint

A universal procedure pointer (UPP) to your component's main entry point. The function referred to by this parameter receives all requests for the component. See the `ComponentRoutineProcPtr` (page 358) callback for more information on creating a component function.

global

A set of flags that control the scope of component registration. See Register Component Resource flags (page 377) for a description of the flags.

componentName

A handle to the component's name. Set this parameter to `NULL` if you do not want to assign a name to the component.

componentInfo

A handle to the component's information string. Set this parameter to `NULL` if you do not want to assign an information string to the component.

componentIcon

A handle to the component's icon (a 32-by-32 pixel black-and-white icon). Set this parameter to `NULL` if you do not want to supply an icon for this component. Note that this icon is not used by the Finder you supply an icon only so that other components or applications can display your component's icon if needed.

Return Value

The unique component identifier assigned to the component by the Component Manager or, if it cannot register the component, `NULL`. See the description of the `Component` data type.

Discussion

Before a component can be used by an application, the component must be registered with the Component Manager. Applications can then find and open the component using the standard Component Manager functions.

Components you register with the `RegisterComponent` function must be in memory when you call this function. If you want to register a component that is stored in the resource fork of a file, use the `RegisterComponentResource` (page 352) function. Use the `RegisterComponentResourceFile` (page 352) function to register all components in the resource fork of a file. The Component Manager automatically registers component resources stored in files with file types of `'thng'` that are stored in the Extensions folder. See "Resources" for more information on component resource files.

Note that a component residing in your application heap remains registered until your application unregisters it or quits. When an application quits, the Component Manager automatically closes any component connections to that application. In addition, if the application has registered components that reside in its heap space, the Component Manager automatically unregisters those components. A component residing in the system heap and registered by your application remains registered until your application unregisters it or until the computer is shut down.

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

Declared In
`Components.h`

RegisterComponentFile

(Deprecated in Mac OS X v10.5.)

```
OSErr RegisterComponentFile (
   const FSSpec *spec,
   short global
);
```

Parameters

spec

Return Value
A result code. See "Component Manager Result Codes" (page 380).

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
`Components.h`

RegisterComponentFileEntries

(Deprecated in Mac OS X v10.5.)

```
OSErr RegisterComponentFileEntries (
   const FSSpec *spec,
   short global,
   const ComponentDescription *toRegister,
   UInt32 registerCount
);
```

Parameters

spec
toRegister
registerCount

Return Value
A result code. See "Component Manager Result Codes" (page 380).

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
`Components.h`

RegisterComponentFileRef

```
OSErr RegisterComponentFileRef (
   const FSRef *ref,
   SInt16 global
);
```

Parameters
ref

Return Value
A result code. See "Component Manager Result Codes" (page 380).

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

Declared In
`Components.h`

RegisterComponentFileRefEntries

```
OSErr RegisterComponentFileRefEntries (
   const FSRef *ref,
   SInt16 global,
   const ComponentDescription *toRegister,
   UInt32 registerCount
);
```

Parameters
ref
toRegister
registerCount

Return Value
A result code. See "Component Manager Result Codes" (page 380).

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

Declared In
`Components.h`

RegisterComponentResource

Registers a component stored in a resource file.

```
Component RegisterComponentResource (
   ComponentResourceHandle cr,
   SInt16 global
);
```

Parameters

cr

A handle to a component resource that describes the component to be registered. The component resource contains all the information required to register the component. Components you register with this function must be stored in a resource file as a component resource. The Component Manager automatically registers component resources stored in files with file types of `'thng'` that are stored in the Extensions folder. See "Resources" for more information on component resource files.

global

A set of flags that controls the scope of component registration. See Register Component Resource flags (page 377) for a description of the flags.

Return Value

The unique component identifier assigned to the component by the Component Manager, or `NULL` if the function could not register the component. See the description of the `Component` data type.

Discussion

Before a component can be used by an application, the component must be registered with the Component Manager. Applications can then find and open the component using the standard Component Manager functions.

If you want to register a component that is in memory, use the `RegisterComponent` (page 348) function.

This function does not actually load the code specified by the component resource into memory. Rather, the Component Manager loads the component code the first time an application opens the component. If the code is not in the same file as the component resource or if the Component Manager cannot find the file, the open request fails.

Note that a component registered locally by your application remains registered until your application unregisters it or quits. When an application quits, the Component Manager automatically closes any component connections to that application. In addition, if the application has registered components that reside in its heap space, the Component Manager automatically unregisters those components. A component registered globally by your application remains registered until your application unregisters it or until the computer is shut down.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

RegisterComponentResourceFile

Registers all component resources in the given resource file.

```
SInt32 RegisterComponentResourceFile (
   SInt16 resRefNum,
   SInt16 global
);
```

Parameters

resRefNum

The reference number of the resource file containing the components to register.

global

A set of flags that control the scope of the registration of the components in the resource file. See Register Component Resource flags (page 377) for a description of the flags.

Return Value

The number of components registered, if all components in the specified resource file are successfully registered. If one or more of the components in the resource file could not be registered, or if the specified file reference number is invalid, a negative function result.

Discussion

Before a component can be used by an application, the component must be registered with the Component Manager. The Component Manager automatically registers component resources stored in files with file types of `'thng'` that are stored in the Extensions folder. For a description of the format and content of component resources, see "Resources".

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

ResolveComponentAlias

```
Component ResolveComponentAlias (
   Component aComponent
);
```

Parameters

aComponent

Return Value

See the description of the `Component` data type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

SetComponentInstanceError

Passes error information to the Component Manager which sets the current error value for the appropriate connection.

```
void SetComponentInstanceError (
   ComponentInstance aComponentInstance,
   OSErr theError
);
```

Parameters

aComponentInstance

The connection for which to set the error. The Component Manager provides a component instance to your component when the connection is opened. The Component Manager also provides a component instance to your component as the first parameter in the `params` field of the parameters structure.

theError

The new value for the current error.

Discussion

In general, your component returns error information in its function result a nonzero function result indicates an error occurred, and a function result of 0 indicates the request was successful. However, some requests require that your component return other information as its function result. In these cases, your component can use this function to report its latest error state to the Component Manager. You can also use this function at any time during your component's execution to report an error.

Applications retrieve this error information by calling the `GetComponentInstanceError` (page 336) function. The documentation for your component should specify how the component indicates errors.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

SetComponentInstanceStorage

Allows your component to associate memory with a connection.

```
void SetComponentInstanceStorage (
   ComponentInstance aComponentInstance,
   Handle theStorage
);
```

Parameters

aComponentInstance

The connection to associate with the allocated memory. The Component Manager provides a component instance to your component when the connection is opened. You can use a component identifier here, but you must coerce the data type appropriately.

theStorage

A handle to the memory that your component has allocated for the connection. Your component must allocate this memory in the current heap. The Component Manager saves this handle and provides it to your component, along with other parameters, in subsequent requests to this connection.

Discussion
When an application or component opens a connection to your component, the Component Manager sends your component an open request. In response to this open request, your component should set up an environment to service the connection. Typically, your component should allocate some memory for the connection. Your component can then use that memory to maintain state information appropriate to the connection.

Your component should dispose of any allocated memory for the connection only in response to the close request. Note that whenever an open request fails, the Component Manager always issues the close request. Furthermore, the value stored with this function is always passed to the close request, so it must be valid or `NULL`. If the open request tries to dispose of its allocated memory before returning, it should call this function again with a `NULL` handle to keep the Component Manager from passing an invalid handle to the close request.

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

Related Sample Code
SoftVDigX

Declared In
`Components.h`

SetComponentRefcon

Sets the reference constant for your component.

```
void SetComponentRefcon (
   Component aComponent,
   long theRefcon
);
```

Parameters

aComponent
The component whose reference constant you wish to set. You can use a component instance here, but you must coerce the data type appropriately.

theRefcon
The reference constant value that you want to set for your component. Your component can retrieve the reference constant using the `GetComponentRefcon` (page 339) function.

Discussion
There is one reference constant for each component, regardless of the number of connections to that component. When your component is registered, the Component Manager sets this reference constant to 0.

The reference constant is a 4-byte value that your component can use in any way you decide. For example, you might use the reference constant to store the address of a data structure that is shared by all connections maintained by your component. You should allocate shared structures in the system heap. Your component should deallocate the structure when its last connection is closed or when it is unregistered.

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

Declared In
`Components.h`

SetDefaultComponent

Changes the search order for registered components.

```
OSErr SetDefaultComponent (
   Component aComponent,
   SInt16 flags
);
```

Parameters

aComponent

The component which you wish moved to the front of the search chain. The order of the search chain influences which component the Component Manager selects in response to an application's use of the `OpenDefaultComponent` (page 348) and `FindNextComponent` (page 333) functions. You can use a component instance here, but you must coerce the data type appropriately.

flags

A value specifying the control information governing the operation. The value of this parameter controls which component description fields the Component Manager examines during the reorder operation. Set the appropriate flags to 1 to define the fields that are examined during the reorder operation. See Set Default Component Flags (page 379) for a description of the values you can use here.

Return Value
A result code. See "Component Manager Result Codes" (page 380).

Discussion
Note that this function changes the search order for all applications. As a result, you should use this function carefully.

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

Declared In
`Components.h`

UncaptureComponent

Allows your component to uncapture a previously captured component.

```
OSErr UncaptureComponent (
   Component aComponent
);
```

Parameters

aComponent

The component to be uncaptured. Your component obtains this identifier from the `CaptureComponent` (page 326) function. You can use a component instance here, but you must coerce the data type appropriately.

Return Value
A result code. See "Component Manager Result Codes" (page 380).

Discussion
This function restores the specified component to the list of available components. Applications can then access the component and retrieve information about the component using Component Manager functions.

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

Declared In
`Components.h`

UnregisterComponent

Removes a component from the Component Manager's registration list.

```
OSErr UnregisterComponent (
   Component aComponent
);
```

Parameters

aComponent

The component to be removed. Applications that register components may obtain this identifier from the `RegisterComponent` (page 348) or `RegisterComponentResource` (page 352) functions. The component must not be in use by any applications or components. You can use a component instance here, but you must coerce the data type appropriately.

Return Value
A result code. See "Component Manager Result Codes" (page 380). If there are open connections to the component, returns a `validInstancesExist` error.

Discussion
Most components are registered at startup and remain registered until the computer is shut down. However, you may want to provide some services temporarily. In that case you dispose of the component that provides the temporary service by using this function.

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

Declared In
`Components.h`

Callbacks

ComponentMPWorkFunctionProcPtr

```
typedef ComponentResult (*ComponentMPWorkFunctionProcPtr) (
   void * globalRefCon,
   ComponentMPWorkFunctionHeaderRecordPtr header
);
```

If you name your function `MyComponentMPWorkFunctionProc`, you would declare it like this:

```
ComponentResult MyComponentMPWorkFunctionProc (
```

```
   void * globalRefCon,
   ComponentMPWorkFunctionHeaderRecordPtr header
);
```

Parameters

header

Return Value
See the description of the `ComponentResult` data type.

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

Declared In
`Components.h`

ComponentRoutineProcPtr

Defines a pointer to your component callback function, which serves as the main entry point into your component and performs the component's services.

```
typedef ComponentResult (*ComponentRoutineProcPtr) (
   ComponentParameters * cp,
   Handle componentStorage
);
```

If you name your function `MyComponentRoutineProc`, you would declare it like this:

```
ComponentResult ComponentRoutineProcPtr (
   ComponentParameters * cp,
   Handle componentStorage
);
```

Parameters

cp

A `ComponentParameters` (page 365) structure. The `what` field of the component parameters structure indicates the action your component should perform. The parameters that the client invoked your function with are contained in the `params` field of the component parameters structure. Your component can use the `CallComponentFunction` (page 321) or `CallComponentFunctionWithStorage` (page 322) function to extract the parameters from this structure.

componentStorage

A handle to any memory that your component has associated with the connection. Typically, upon receiving an open request, your component allocates memory and uses the `SetComponentInstanceStorage` (page 354) function to associate the allocated memory with the component connection.

Return Value
Your component should return a value of type `ComponentResult`. If your component does not return error information as its function result, it should indicate errors using the `SetComponentInstanceError` (page 353) function. See the description of the `ComponentResult` data type.

Discussion
You pass a pointer to your component callback function to the Component Manager when you register your component. The Component Manager can then call your component when another application or component requests its services. When your component receives a request, it should perform the action specified in the `what` field of the component parameters structure.

The pointer which you pass to the Component Manager should be a universal procedure pointer (UPP). The definition of the UPP data type for your component function is as follows:

```
typedef (ComponentRoutineProcPtr) ComponentRoutineUPP;
```

Before using your component function, you must first create a UPP for your callback function, using the `NewComponentRoutineUPP` (page 344) function, as shown here:

```
ComponentRoutineUPP MyComponentRoutineUPP;
MyComponentRoutineUPP =
NewComponentRoutineUPP(&MyComponentRoutineProc)
```

You then pass `MyComponentRoutineUPP` to the Component Manager when you register your component. The Component Manager will call your function each time your component receives a request. If you wish to call your component function yourself, you can use the `InvokeComponentRoutineUPP` (page 342) function.

```
result = InvokeComponentRoutineUPP &myParams, myStorage,
MyComponentRoutineUPP)
```

When you are finished with your component callback function, you should dispose of the universal procedure pointer associated with it, using the `DisposeComponentRoutineUPP` (page 333) function.

```
DisposeComponentRoutineUPP(MyComponentRoutineUPP);
```

To provide a component, you define a component function and supply the appropriate registration information. You store your component function in a code resource and typically store your component's registration information as resources in a component file.

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

Declared In
`Components.h`

GetMissingComponentResourceProcPtr

```
typedef OSErr (*GetMissingComponentResourceProcPtr) (
   Component c,
   OSType resType,
   short resID,
   void * refCon,
   Handle * resource
);
```

If you name your function `MyGetMissingComponentResourceProc`, you would declare it like this:

```
OSErr GetMissingComponentResourceProcPtr (
   Component c,
   OSType resType,
```

```
    short resID,
    void * refCon,
    Handle * resource
);
```

Parameters

c

resType

resource

Return Value
A result code. See "Component Manager Result Codes" (page 380).

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

Declared In
`Components.h`

Data Types

ComponentAliasResource

```
struct ComponentAliasResource {
    ComponentResource cr;
    ComponentDescription aliasCD;
};
typedef struct ComponentAliasResource ComponentAliasResource;
```

Fields

`cr`

`aliasCD`

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

Declared In
`Components.h`

ComponentDependencyArray

```
struct ComponentDependencyArray {
    SInt32 count;
    ComponentDescription descArray[1];
};
```

Fields

`count`

`descArray`

ComponentDescription

```
struct ComponentDescription {
    OSType componentType;
    OSType componentSubType;
    OSType componentManufacturer;
    unsigned long componentFlags;
    unsigned long componentFlagsMask;
};
typedef struct ComponentDescription ComponentDescription;
```

Fields

`componentType`

A four-character code that identifies the type of component. All components of a particular type must support a common set of interface functions. For example, drawing components all have a component type of `'draw'`.

If you are developing an application which uses components, you can use this field to search for components of a given type by specifying the component type in this field of the component description structure you supply to the `FindNextComponent` (page 333) function or the `CountComponents` (page 330) function. A value of 0 operates as a wildcard.

If you are developing a component, it must support all of the standard functions for the component type specified by this field. Type codes with all lowercase characters are reserved for definition by Apple. You can define your own component type code as long as you register it with Apple's Component Registry Group.

`componentSubType`

A four-character code that identifies the subtype of the component. Different subtypes of a component type may support additional features or provide interfaces that extend beyond the standard functions for a given component type. For example, the subtype of drawing components indicates the type of object the component draws. Drawing components that draw ovals have a subtype of `'oval'`.

If you are developing an application which uses components, you can use the `componentSubType` field to perform a more specific lookup operation than is possible using only the `componentType` field. By specifying particular values for both fields in the component description structure that you supply to the `FindNextComponent` or `CountComponents` function, your application retrieves information about only those components that meet both of these search criteria. A value of 0 operates as a wildcard.

If you are developing a component, you may use this field to indicate more specific information about the capabilities of the component. There are no restrictions on the content you assign to this field. If no additional information is appropriate for your component type, you may set the `componentSubType` field to 0.

`componentManufacturer`

A four-character code that identifies the manufacturer of the component. This field allows for further differentiation between individual components. For example, components made by a specific manufacturer may support an extended feature set. Components provided by Apple use a manufacturer value of `'appl'`.

If you are developing an application which uses components, you can use this field to find components from a certain manufacturer. Specify the appropriate manufacturer code in this field of the component description structure you supply to the `FindNextComponent` or `CountComponents` function. A value of 0 operates as a wildcard.

If you are developing a component, you obtain your manufacturer code, which can be the same as your application signature, from Apple's Component Registry Group.

`componentFlags`

A 32-bit field that provides additional information about a particular component.

The high-order 8 bits are reserved for definition by the Component Manager. If you are developing an application, you should usually set these bits to 0.

The low-order 24 bits are specific to each component type. These flags can be used to indicate the presence of features or capabilities in a given component.

If you are developing an application which uses components, you can use these flags to further narrow the search criteria applied by the `FindNextComponent` or `CountComponents` function. If you use the `componentFlags` field in a component search, you use the `componentFlagsMask` field to indicate which flags are to be considered in the search.

If you are developing a component, you can use these flags to indicate any special capabilities or features of your component. You may use all 24 bits, as appropriate to its component type. You must set all unused bits to 0.

`componentFlagsMask`

A 32-bit field that indicates which flags in the `componentFlags` field are relevant to a particular component search operation.

If you are developing an application which uses components, your application should set each bit which corresponds to a flag in the `componentFlags` field that is to be considered as a search criterion by the `FindNextComponent` or `CountComponents` function to 1. The Component Manager considers only these flags during the search. You specify the desired flag value (either 0 or 1) in the `componentFlags` field.

For example, to look for a component with a specific control flag that is set to 0, set the appropriate bit in the `ComponentFlags` field to 0 and the same bit in the `ComponentFlagsMask` field to 1. To look for a component with a specific control flag that is set to 1, set the bit in the `ComponentFlags` field to 1 and the same bit in the `ComponentFlagsMask` field to 1. To ignore a flag, set the bit in the `ComponentFlagsMask` field to 0.

If you are developing a component, your component must set the `componentFlagsMask` field in its component description structure to 0.

Discussion

The ComponentDescription structure identifies the characteristics of a component, including the type of services offered by the component and its manufacturer.

Applications and components use component description structures in different ways. An application that uses components specifies the selection criteria for a component in a component description structure. The functions `FindNextComponent` (page 333) , `CountComponents` (page 330) , and `GetComponentInfo` (page 335) all use the component description structure to specify the criteria for their search.

A component uses the component description structure to specify its registration information and capabilities and identify itself to the Component Manager. If your component is stored in a component resource, the information in the component description structure must be part of that resource. See the description of the component 'thng' resource. If you have developed an application that registers your component, that application must supply a component description structure to the `RegisterComponent` (page 348) function. See "Registering Components" for information about registering components.

The `ComponentDescription` data type defines the component description structure. Note that the valid values of fields in the component description structure are determined by the component type specification. For example, all image compressor components must use the `componentSubType` field to specify the compression algorithm used by the compressor.

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

Declared In
`Components.h`

ComponentFunctionUPP

```
typedef UniversalProcPtr ComponentFunctionUPP;
```

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

Declared In
`Components.h`

ComponentInstanceRecord

```
struct ComponentInstanceRecord {
    long data[1];
};
typedef struct ComponentInstanceRecord ComponentInstanceRecord;
typedef ComponentInstanceRecord * ComponentInstance;
```

Fields
`data`

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

Declared In
`Components.h`

ComponentMPWorkFunctionHeaderRecord

```
struct ComponentMPWorkFunctionHeaderRecord {
    UInt32 headerSize;
    UInt32 recordSize;
    UInt32 workFlags;
    UInt16 processorCount;
    UInt8 unused;
    UInt8 isRunning;
};
typedef struct ComponentMPWorkFunctionHeaderRecord
ComponentMPWorkFunctionHeaderRecord;
typedef ComponentMPWorkFunctionHeaderRecord *
ComponentMPWorkFunctionHeaderRecordPtr;
```

Fields

`headerSize`

`recordSize`

`workFlags`

`processorCount`

`unused`

`isRunning`

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

Declared In
`Components.h`

ComponentMPWorkFunctionUPP

```
typedef ComponentMPWorkFunctionProcPtr ComponentMPWorkFunctionUPP;
```

Discussion
For more information, see the description of the ComponentMPWorkFunctionUPP callback function.

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

Declared In
`Components.h`

ComponentParameters

```
struct ComponentParameters {
    UInt8 flags;
    UInt8 paramSize;
    short what;
    long params[1];
};
typedef struct ComponentParameters ComponentParameters;
```

Fields

`flags`

Reserved for use by Apple.

`paramSize`

Specifies the number of bytes of parameter data for this request. The actual parameters are stored in the `params` field.

`what`

Specifies the type of request. Component designers define the meaning of positive values and assign them to requests that are supported by components of a given type. Negative values are reserved for definition by Apple. See "Result Codes" for Apple-defined request code values.

`params`

An array that contains the parameters specified by the application that called your component. You can use the `CallComponentRoutine` or `CallComponentRoutineWithStorage` function to convert this array into a Pascal-style invocation of a subroutine in your component.

Discussion

The Component Manager uses the component parameters structure to pass information to your component about a request from an application. Functions which use this data type are `CallComponentFunction` (page 321), `CallComponentFunctionWithStorage` (page 322), and `DelegateComponentCall` (page 331).The information in this structure completely defines the request. Your component services the request as appropriate.

The `ComponentParameters` data type defines the component parameters structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Components.h`

ComponentPlatformInfo

```
struct ComponentPlatformInfo {
    long componentFlags;
    ResourceSpec component;
    short platformType;
};
typedef struct ComponentPlatformInfo ComponentPlatformInfo;
```

Fields

`component`

Availability

Available in Mac OS X v10.0 and later.

Declared In
Components.h

ComponentPlatformInfoArray

```
struct ComponentPlatformInfoArray {
    long count;
    ComponentPlatformInfo platformArray[1];
};
typedef struct ComponentPlatformInfoArray ComponentPlatformInfoArray;
```

Fields
platformArray

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

Declared In
Components.h

ComponentRecord

```
struct ComponentRecord {
    long data[1];
};
typedef struct ComponentRecord ComponentRecord;
typedef ComponentRecord * Component;
```

Fields
data

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

Declared In
Components.h

ComponentResource

```
struct ComponentResource {
    ComponentDescription cd;
    ResourceSpec component;
    ResourceSpec componentName;
    ResourceSpec componentInfo;
    ResourceSpec componentIcon;
};
typedef struct ComponentResource ComponentResource;
typedef ComponentResource * ComponentResourcePtr;
```

Fields
cd

A ComponentDescription (page 361) structure that specifies the characteristics of the component.

`component`
: A resource specification structure that specifies the type and ID of the component code resource. The `resType` field of the resource specification structure may contain any value. The component's main entry point must be at offset 0 in the resource.

`componentName`
: A resource specification structure that specifies the resource type and ID for the name of the component. This is a Pascal string. Typically, the name is stored in a resource of type `'STR '`.

`componentInfo`
: A resource specification structure that specifies the resource type and ID for the information string that describes the component. This is a Pascal string. Typically, the information string is stored in a resource of type `'STR '`. You might use the information stored in this resource in a Get Info dialog box.

`componentIcon`
: A resource specification structure that specifies the resource type and ID for the icon for a component. Component icons are stored as 32-by-32 bit maps. Typically, the icon is stored in a resource of type `'ICON'`. Note that this icon is not used by the Finder you supply an icon only so that other components or applications can display your component's icon in a dialog box if needed.

Discussion
The `ComponentResource` data type defines the structure of a component resource. You can also optionally append to the end of this structure the information defined by the `ComponentResourceExtension` (page 367) data type.

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

Declared In
`Components.h`

ComponentResourceExtension

```
struct ComponentResourceExtension {
    long componentVersion;
    long componentRegisterFlags;
    short componentIconFamily;
};
typedef struct ComponentResourceExtension ComponentResourceExtension;
```

Fields
`componentVersion`
: The version number of the component. If you specify the `componentDoAutoVersion` flag in `componentRegisterFlags`, the Component Manager must obtain the version number of your component when your component is registered. Either you can provide a version number in your component's resource, or you can specify a value of 0 for its version number. If you specify 0, the Component Manager sends your component a version request to get the version number of your component.

`componentRegisterFlags`
: A set of flags containing additional registration information. See Component Resource Extension Flags (page 372) for the flag values.

`componentIconFamily`
: The resource ID of an icon family. You 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 a dialog box if needed.

Discussion
You can optionally include in your component resource the information defined by the `ComponentResourceExtension` data type:

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

Declared In
`Components.h`

ComponentResult

```
typedef long ComponentResult;
```

Discussion

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

Declared In
`Components.h`

ComponentRoutineUPP

```
typedef ComponentRoutineProcPtr ComponentRoutineUPP;
```

Discussion
For more information, see the description of the ComponentRoutineUPP callback function.

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

Declared In
`Components.h`

ExtComponentResource

```
struct ExtComponentResource {
    ComponentDescription cd;
    ResourceSpec component;
    ResourceSpec componentName;
    ResourceSpec componentInfo;
    ResourceSpec componentIcon;
    long componentVersion;
    long componentRegisterFlags;
    short componentIconFamily;
    long count;
    ComponentPlatformInfo platformArray[1];
};
typedef struct ExtComponentResource ExtComponentResource;
typedef ExtComponentResource * ExtComponentResourcePtr;
```

Fields

`cd`

`component`

`componentName`

`componentInfo`

`componentIcon`

`platformArray`

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

Declared In
`Components.h`

GetMissingComponentResourceUPP

```
typedef GetMissingComponentResourceProcPtr GetMissingComponentResourceUPP;
```

Discussion
For more information, see the description of the GetMissingComponentResourceUPP callback function.

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

Declared In
`Components.h`

RegisteredComponentInstanceRecord

```
struct RegisteredComponentInstanceRecord {
    long data[1];
};
typedef struct RegisteredComponentInstanceRecord  RegisteredComponentInstanceRecord;
typedef RegisteredComponentInstanceRecord *  RegisteredComponentInstanceRecordPtr;
```

Fields
`data`

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

Declared In
`Components.h`

RegisteredComponentRecord

```
struct RegisteredComponentRecord {
    long data[1];
};
typedef struct RegisteredComponentRecord RegisteredComponentRecord;
typedef RegisteredComponentRecord * RegisteredComponentRecordPtr;
```

Fields
`data`

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

Declared In
`Components.h`

ResourceSpec

```
struct ResourceSpec {
    OSType resType;
    short resID;
};
typedef struct ResourceSpec ResourceSpec;
```

Fields
`resType`
The type of the resource.

`resID`
The ID of the resource.

Discussion
The `ComponentResource` (page 366) structure uses the resource specification structure, defined by the `ResourceSpec` data type, to describe the component's code, name, information string, or icon. The resources specified by the resource specification structures must reside in the same resource file as the component resource itself.

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

Declared In
`Components.h`

Constants

cmpAliasNoFlags

```
enum {
    cmpAliasNoFlags = 0,
    cmpAliasOnlyThisFile = 1
};
```

Constants
`cmpAliasNoFlags`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`cmpAliasOnlyThisFile`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

cmpIsMissing

```
enum {
    cmpThreadSafe = 1L << 28,
    cmpIsMissing = 1L << 29,
    cmpWantsRegisterMessage = 1L << 31
};
```

Constants
`cmpThreadSafe`
Available in Mac OS X v10.3 and later.
Declared in `Components.h`.

`cmpIsMissing`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`cmpWantsRegisterMessage`
The setting of the `cmpWantsRegisterMessage` bit determines whether the Component Manager calls this component during registration. Set this bit to 1 if your component should be called when it is registered; otherwise, set this bit to 0.
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

Discussion
These values are used by the `componentFlags` field of the `ComponentDescription` (page 361) structure to provide additional information about a component.

Component Resource Extension Flags

```
enum {
    componentDoAutoVersion = (1 << 0),
    componentWantsUnregister = (1 << 1),
    componentAutoVersionIncludeFlags = (1 << 2),
    componentHasMultiplePlatforms = (1 << 3),
    componentLoadResident = (1 << 4)
};
```

Constants

`componentDoAutoVersion`
Specify this flag if you want the Component Manager to resolve conflicts between different versions of the same component. If you specify this flag, the Component Manager registers your component only if there is no later version available. If an older version is already registered, the Component Manager unregisters it. If a newer version of the same component is registered after yours, the Component Manager automatically unregisters your component. You can use this automatic version control feature to make sure that the most recent version of your component is registered, regardless of the number of versions that are installed

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`componentWantsUnregister`
Specify this flag if you want your component to receive an unregister request when it is unregistered.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`componentAutoVersionIncludeFlags`
Specify this flag if you want the Component Manager to include the `componentFlags` field of the component description structure when it searches for identical components in the process of performing automatic version control for your component. If you do not specify this flag, the Component Manager searches only the `componentType`, `componentSubType`, and `componentManufacturer` fields.

Note that the setting of the `componentAutoVersionIncludeFlags` flag affects automatic version control only and does not affect the search operations performed by `FindNextComponent` and `CountComponents`.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`componentHasMultiplePlatforms`
Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`componentLoadResident`
Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

Discussion
These values are used in the `ComponentResourceExtension` (page 367) structure to specify additional information about component registration.

CSComponentsThreadMode

```
typedef UInt32 CSComponentsThreadMode;
enum {
    kCSAcceptAllComponentsMode = 0,
    kCSAcceptThreadSafeComponentsOnlyMode = 1
};
```

Constants

`kCSAcceptAllComponentsMode`
Available in Mac OS X v10.3 and later.
Declared in `Components.h`.

`kCSAcceptThreadSafeComponentsOnlyMode`
Available in Mac OS X v10.3 and later.
Declared in `Components.h`.

kAnyComponentType

```
enum {
    kAnyComponentType = 0,
    kAnyComponentSubType = 0,
    kAnyComponentManufacturer = 0,
    kAnyComponentFlagsMask = 0
};
```

Constants

`kAnyComponentType`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`kAnyComponentSubType`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`kAnyComponentManufacturer`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`kAnyComponentFlagsMask`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

kAppleManufacturer

```
enum {
    kAppleManufacturer = 'appl',
    kComponentResourceType = 'thng',
    kComponentAliasResourceType = 'thga'
};
```

Constants

`kAppleManufacturer`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`kComponentResourceType`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`kComponentAliasResourceType`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

mpWorkFlagDoWork

```
enum {
    mpWorkFlagDoWork = (1 << 0),
    mpWorkFlagDoCompletion = (1 << 1),
    mpWorkFlagCopyWorkBlock = (1 << 2),
    mpWorkFlagDontBlock = (1 << 3),
    mpWorkFlagGetProcessorCount = (1 << 4),
    mpWorkFlagGetIsRunning = (1 << 6)
};
```

Constants

`mpWorkFlagDoWork`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`mpWorkFlagDoCompletion`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`mpWorkFlagCopyWorkBlock`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`mpWorkFlagDontBlock`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`mpWorkFlagGetProcessorCount`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`mpWorkFlagGetIsRunning`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

platform68k

```
enum {
    platform68k = 1,
    platformPowerPC = 2,
    platformInterpreted = 3,
    platformWin32 = 4,
    platformPowerPCNativeEntryPoint = 5,
    platformIA32NativeEntryPoint = 6
};
```

Constants

`platform68k`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformPowerPC`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformInterpreted`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformWin32`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformPowerPCNativeEntryPoint`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformIA32NativeEntryPoint`
Available in Mac OS X v10.3 and later.
Declared in `Components.h`.

Discussion

platformIRIXmips

```
enum {
    platformIRIXmips = 1000,
    platformSunOSsparc = 1100,
    platformSunOSintel = 1101,
    platformLinuxppc = 1200,
    platformLinuxintel = 1201,
    platformAIXppc = 1300,
    platformNeXTIntel = 1400,
    platformNeXTppc = 1401,
    platformNeXTsparc = 1402,
    platformNeXT68k = 1403,
    platformMacOSx86 = 1500
};
```

Constants

`platformIRIXmips`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformSunOSsparc`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformSunOSintel`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformLinuxppc`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformLinuxintel`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformAIXppc`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformNeXTIntel`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformNeXTppc`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformNeXTsparc`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformNeXT68k`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`platformMacOSx86`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

Register Component Resource flags

```
enum {
    registerComponentGlobal = 1,
    registerComponentNoDuplicates = 2,
    registerComponentAfterExisting = 4,
    registerComponentAliasesOnly = 8
};
```

Constants

`registerComponentGlobal`
Specify this flag to indicate that this component should be made available to other applications and clients as well as the one performing the registration. If you do not specify this flag, the component is available for use only by the registering application or component (that is, the component is local to the A5 world of the registering program).
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`registerComponentNoDuplicates`
Specify this flag to indicate that if a component with identical characteristics to the one being registered already exists, then the new one should not be registered (`RegisterComponent` returns 0 in this situation). If you do not specify this flag, the component is registered even if a component with identical characteristics to the one being registered already exists.
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`registerComponentAfterExisting`
Specify this flag to indicate that this component should be registered after all other components with the same component type. Usually components are registered before others with identical descriptions; specifying this flag overrides that behavior.
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

`registerComponentAliasesOnly`
Available in Mac OS X v10.0 and later.
Declared in `Components.h`.

Discussion

The functions `RegisterComponent` (page 348) , `RegisterComponentResource` (page 352) , and `RegisterComponentResourceFile` (page 352) use these flags in the `global` parameter.

Request Codes

```
enum {
    kComponentOpenSelect = -1,
    kComponentCloseSelect = -2,
    kComponentCanDoSelect = -3,
    kComponentVersionSelect = -4,
    kComponentRegisterSelect = -5,
    kComponentTargetSelect = -6,
    kComponentUnregisterSelect = -7,
    kComponentGetMPWorkFunctionSelect = -8,
    kComponentExecuteWiredActionSelect = -9,
    kComponentGetPublicResourceSelect = -10
};
```

Constants

`kComponentOpenSelect`

A request to open a connection. Your component must respond to this request code.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentCloseSelect`

A request to close a connection. Your component must respond to this request code.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentCanDoSelect`

A request to determine whether your component supports a particular request. Your component must respond to this request code

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentVersionSelect`

A request to return your component's version number. Your component must respond to this request code.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentRegisterSelect`

A request to determine whether your component can operate in the current environment. Your component may or may not respond to this request code.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentTargetSelect`

A request to call another component whenever your component would call itself. Your component may or may not respond to this request code.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentUnregisterSelect`
A request to perform any operations necessary as a result of your component being unregistered. Your component may or may not respond to this request code

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentGetMPWorkFunctionSelect`
Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentExecuteWiredActionSelect`
Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`kComponentGetPublicResourceSelect`
Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

Discussion
These values are used in the `ComponentParameters` (page 365) structure to specify the type of a request to a component. Apple has defined these request codes:

Set Default Component Flags

```
enum {
    defaultComponentIdentical = 0,
    defaultComponentAnyFlags = 1,
    defaultComponentAnyManufacturer = 2,
    defaultComponentAnySubType = 4,
    defaultComponentAnyFlagsAnyManufacturer = (defaultComponentAnyFlags  +
defaultComponentAnyManufacturer),
    defaultComponentAnyFlagsAnyManufacturerAnySubType =  (defaultComponentAnyFlags
 + defaultComponentAnyManufacturer +  defaultComponentAnySubType)
};
```

Constants
`defaultComponentIdentical`
The Component Manager places the component specified in the call to `SetDefaultComponent` in front of all other components that have the same component description.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`defaultComponentAnyFlags`
The Component Manager ignores the value of the `componentFlags` field during the reorder operation.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`defaultComponentAnyManufacturer`
The Component Manager ignores the value of the `componentManufacturer` field during the reorder operation.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`defaultComponentAnySubType`
The Component Manager ignores the value of the `componentSubType` field during the reorder operation.

Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`defaultComponentAnyFlagsAnyManufacturer`
Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

`defaultComponentAnyFlagsAnyManufacturerAnySubType`
Available in Mac OS X v10.0 and later.

Declared in `Components.h`.

Discussion
The `SetDefaultComponent` (page 356) function uses these values in the `flags` parameter to control which component description fields the Component Manager examines during the reorder operation.

Result Codes

The result codes defined by the Component Manager are listed below.

Result Code	Value	Description
`invalidComponentID`	-3000	Invalid component ID. Available in Mac OS X v10.0 and later.
`validInstancesExist`	-3001	This component has open connections. Available in Mac OS X v10.0 and later.
`componentNotCaptured`	-3002	This component has not been captured. Available in Mac OS X v10.0 and later.
`componentDontRegister`	-3003	Available in Mac OS X v10.0 and later.
`unresolvedComponentDLLErr`	-3004	Available in Mac OS X v10.0 and later.
`retryComponentRegistrationErr`	-3005	Available in Mac OS X v10.0 and later.
`badComponentSelector`	0x80008002	Component does not support the specified request code. Available in Mac OS X v10.0 and later.
`badComponentInstance`	0x80008001	Invalid component passed to Component Manager. Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availability information by using the Component Manager selectors defined in the Gestalt Manager. For more information, see *Gestalt Manager Reference*.

Date, Time, and Measurement Utilities Reference

Framework: CoreServices/CoreServices.h

Declared in DateTimeUtils.h
UTCUtils.h

Overview

You can use the Date, Time, and Measurement Utilities to manipulate the date-time information and geographic location data used by a Macintosh computer. A Macintosh computer contains a battery-operated clock chip that maintains information on the current date-time. In Mac OS 9, the date and time were retrieved from this clock chip. In Mac OS X, the date and time are retrieved through the BSD/CoreFoundation level.

You can use the routines provided by the Date, Time, and Measurement Utilities to

- get the current date and time
- set the current date and time, if necessary
- convert between internal date-time structures
- get and set the geographic location and time-zone information
- determine the number of elapsed microseconds since system startup

Please note, setting the time or geographical location requires authorization by using Authorization Services. See Authorization Concepts for more information.

To make the best use of the Date, Time, and Measurement Utilities, you should be familiar with the international resources, especially the numeric-format and long-date-format resources, and the Script Manager.

Carbon supports the majority of the Date, Time, and Measurement Utilities. However, obsolete functions that are prefixed with “iu” or “IU” (such as `IUDateString` and `IUTimeString`) are not supported.

Functions by Task

Converting Between Date-Time Formats

`DateToSeconds` (page 390) Deprecated in Mac OS X v10.3
Converts a date and time to a number of seconds elapsed since midnight, January 1, 1904. (Deprecated. Use the `CFCalendarRef` data type and the functions that operate on it instead.)

SecondsToDate (page 397) Deprecated in Mac OS X v10.3
Converts a number of seconds elapsed since midnight, January 1, 1904 to a date and time. (Deprecated. Use the CFCalendarRef data type and the functions that operate on it instead.)

Converting Between Long Date-Time Format

LongDateToSeconds (page 395) Deprecated in Mac OS X v10.3
Converts a date and time to the number of seconds elapsed since midnight, January 1, 1904. (Deprecated. Use the CFCalendarRef data type and the functions that operate on it instead.)

LongSecondsToDate (page 395) Deprecated in Mac OS X v10.3
Converts the number of seconds elapsed since midnight, January 1, 1904 to a date and time. (Deprecated. Use the CFCalendarRef data type and the functions that operate on it instead.)

Converting Date and Time Strings Into Numeric Representations

InitDateCache (page 393) Deprecated in Mac OS X v10.3
Initializes the date cache structure, which is used to store data for use by the StringToDate and StringToTime functions. (Deprecated. There is no replacement.)

StringToDate (page 400) Deprecated in Mac OS X v10.3
Parses a string for a date and converts the date information into values in a date-time structure. (Deprecated. Use CFDateFormatterCreateDateFromString instead.)

StringToTime (page 401) Deprecated in Mac OS X v10.3
Parses a string for a time specification and converts the date information into values in a date-time structure. (Deprecated. Use CFDateFormatterCreateDateFromString instead.)

Converting Long Date and Time Values Into Strings

LongDateString (page 394) Deprecated in Mac OS X v10.3
Converts a date that is specified as a LongDateTime value into a Pascal string, making use of the date formatting information in the specified resource. (Deprecated. Use CFDateFormatterCreateStringWithDate instead.)

LongTimeString (page 396) Deprecated in Mac OS X v10.3
Converts a time that is specified as a LongDateTime value into a Pascal string, making use of the time formatting information in the specified resource. (Deprecated. Use CFDateFormatterCreateStringWithDate instead.)

Converting Numeric Representations Into Date and Time Strings

DateString (page 389) Deprecated in Mac OS X v10.3
Converts a date in the standard date-time representation into a Pascal string, making use of the date formatting information in the specified resource. (Deprecated. Use CFDateFormatterCreateStringWithDate instead.)

`TimeString` (page 402) Deprecated in Mac OS X v10.3
Converts a time in the standard date-time representation into a string, making use of the time formatting information in the specified resource. (Deprecated. Use `CFDateFormatterCreateStringWithDate` instead.)

Converting Between CF and Carbon Time Types

`UCConvertCFAbsoluteTimeToUTCDateTime` (page 406)
Converts a value of type `CFAbsoluteTime` to `UTCDateTime`.

`UCConvertCFAbsoluteTimeToSeconds` (page 405)
Converts a value of type `CFAbsoluteTime` to seconds.

`UCConvertCFAbsoluteTimeToLongDateTime` (page 405)
Converts a value of type `CFAbsoluteTime` to `LongDateTime`.

`UCConvertLongDateTimeToCFAbsoluteTime` (page 406)
Converts a value of type `LongDateTime` to `CFAbsoluteTime`.

`UCConvertSecondsToCFAbsoluteTime` (page 407)
Converts a value from the normal seconds time representation to `CFAbsoluteTime`.

`UCConvertUTCDateTimeToCFAbsoluteTime` (page 408)
Converts a value of type `UTCDateTime` time to `CFAbsoluteTime`.

Converting Between UTC and Local Time

`ConvertLocalTimeToUTC` (page 386) Deprecated in Mac OS X v10.4
Converts local time to UTC. (Deprecated. Use `CFTimeZoneGetSecondsFromGMT` instead.)

`ConvertLocalToUTCDateTime` (page 387) Deprecated in Mac OS X v10.4
Converts local date and time to UTC date and time. (Deprecated. Use `CFTimeZoneGetSecondsFromGMT` instead.)

`ConvertUTCToLocalDateTime` (page 388) Deprecated in Mac OS X v10.4
Converts UTC date and time to local date and time. (Deprecated. Use `CFTimeZoneGetSecondsFromGMT` instead.)

`ConvertUTCToLocalTime` (page 388) Deprecated in Mac OS X v10.4
Converts UTC time to local time. (Deprecated. Use `CFTimeZoneGetSecondsFromGMT` instead.)

Getting the Current Date and Time

`GetTime` (page 392)
Obtains the current date-time information, expressed as a date and time. (Deprecated. Use `CFAbsoluteTimeGetCurrent` instead.)

`GetDateTime` (page 390) Deprecated in Mac OS X v10.3
Obtains the current date-time information, expressed as the number of seconds elapsed since midnight, January 1, 1904. (Deprecated. Use `CFAbsoluteTimeGetCurrent` instead.)

`ReadDateTime` (page 397) Deprecated in Mac OS X v10.3
Reads time information from the system. (Deprecated. Use `CFAbsoluteTimeGetCurrent` instead.)

GetLocalDateTime (page 391) Deprecated in Mac OS X v10.4
Gets the local date and time. (Deprecated. Use CFAbsoluteTimeGetCurrent and CFTimeZoneGetSecondsFromGMT instead.)

GetUTCDateTime (page 392) Deprecated in Mac OS X v10.4
Gets the UTC date and time. (Deprecated. Use CFAbsoluteTimeGetCurrent instead.)

Modifying and Verifying Long Date-Time Records

ToggleDate (page 403) Deprecated in Mac OS X v10.3
Modifies a date and time, by modifying one specific component of a date and time (day, hour, minute, seconds, day of week, and so on). (Deprecated. Use the CFCalendarRef data type and the functions that operate on it instead.)

ValidDate (page 408) Deprecated in Mac OS X v10.3
Verifies specific date and time values in a long date-time structure. (Deprecated. Use the CFCalendarRef data type and the functions that operate on it instead.)

Setting the Current Date and Time

SetDateTime (page 398) Deprecated in Mac OS X v10.3
Changes the date-time information stored by the system to the specified value, expressed as the number of seconds elapsed since midnight, January 1, 1904. (Deprecated. There is no replacement.)

SetTime (page 399) Deprecated in Mac OS X v10.3
Changes the date-time information in the system to the specified value, expressed as a date and time. (Deprecated. There is no replacement.)

SetLocalDateTime (page 398) Deprecated in Mac OS X v10.4
Sets the local date and time. (Deprecated. There is no replacement.)

SetUTCDateTime (page 400) Deprecated in Mac OS X v10.4
Sets the UTC date and time. (Deprecated. Use settimeofday(2) instead.)

Functions

ConvertLocalTimeToUTC

Converts local time to UTC. (Deprecated in Mac OS X v10.4. Use CFTimeZoneGetSecondsFromGMT instead.)

```
OSStatus ConvertLocalTimeToUTC (
   UInt32 localSeconds,
   UInt32 *utcSeconds
);
```

Parameters

localSeconds
A value of type UInt32 containing the local time.

utcSeconds
A pointer to a value of type `UInt32`. On return, this points to the UTC value corresponding to the given time in `localSeconds`.

Return Value
A result code. See "Date, Time, and Measurement Utilities Result Codes" (page 421).

Discussion
Given a local time in `localSeconds`, the function will place the corresponding UTC value in `utcSeconds`. This function returns `noErr` if the conversion is successful. Otherwise, it may return `kUTCUnderflowErr` or `kUTCOverflowErr`.

Special Considerations

For information on using `CFTimeZoneGetSecondsFromGMT`, see *Dates and Times Programming Guide for Core Foundation* and*CFTimeZone 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
`UTCUtils.h`

ConvertLocalToUTCDateTime

Converts local date and time to UTC date and time. (Deprecated in Mac OS X v10.4. Use `CFTimeZoneGetSecondsFromGMT` instead.)

```
OSStatus ConvertLocalToUTCDateTime (
   const LocalDateTime *localDateTime,
   UTCDateTime *utcDateTime
);
```

Parameters

localDateTime
A value of type `LocalDateTime` containing the local date and time.

utcDateTime
A pointer to a value of type `UTCDateTime`. On return, this points to the UTC value corresponding to the given date and time in `localDateTime`.

Return Value
A result code. See "Date, Time, and Measurement Utilities Result Codes" (page 421).

Discussion
Given a local date and time in the `localDateTime` parameter, this function places the corresponding UTC value in `utcDateTime`. This function returns `noErr` if the conversion is successful. Otherwise, it may return `kUTCUnderflowErr`, `kUTCOverflowErr`, or `paramErr` if `utcDateTime` is `NULL`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`UTCUtils.h`

ConvertUTCToLocalDateTime

Converts UTC date and time to local date and time. (Deprecated in Mac OS X v10.4. Use `CFTimeZoneGetSecondsFromGMT` instead.)

```
OSStatus ConvertUTCToLocalDateTime (
   const UTCDateTime *utcDateTime,
   LocalDateTime *localDateTime
);
```

Parameters

`utcDateTime`
A value of type `UTCDateTime` specifying the UTC date and time.

`localDateTime`
A pointer to a value of type `LocalDateTime`. On return, this points to the local value corresponding to the given date and time in `utcDateTime`.

Return Value
A result code. See "Date, Time, and Measurement Utilities Result Codes" (page 421).

Discussion
Given a UTC date and time in `utcDateTime`, this function places the corresponding local value in `localDateTime`. This function returns `noErr` if the conversion is successful. Otherwise, it may return `kUTCUnderflowErr`, `kUTCOverflowErr`, or `paramErr` if `localDateTime` is `NULL`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`UTCUtils.h`

ConvertUTCToLocalTime

Converts UTC time to local time. (Deprecated in Mac OS X v10.4. Use `CFTimeZoneGetSecondsFromGMT` instead.)

```
OSStatus ConvertUTCToLocalTime (
   UInt32 utcSeconds,
   UInt32 *localSeconds
);
```

Parameters

`utcSeconds`
A value of type `UInt32` specifying UTC time in seconds.

`localSeconds`
A pointer to a value of type `UInt32`. On return, this points to the local time corresponding to the UTC time specified in `utcSeconds`.

Return Value
A result code. See "Date, Time, and Measurement Utilities Result Codes" (page 421).

Discussion
Given a UTC time in `utcSeconds` this function places the corresponding local value in `localSeconds`. This function returns `noErr` if the conversion is successful. Otherwise, it may return `kUTCUnderflowErr` or `kUTCOverflowErr`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`UTCUtils.h`

DateString

Converts a date in the standard date-time representation into a Pascal string, making use of the date formatting information in the specified resource. (Deprecated in Mac OS X v10.3. Use `CFDateFormatterCreateStringWithDate` instead.)

```
void DateString (
   SInt32 dateTime,
   DateForm longFlag,
   Str255 result,
   Handle intlHandle
);
```

Parameters

`dateTime`

The date-time value in the representation returned by the `GetDateTime` function. The numeric representation used in these functions is the standard date-time representation: a 32-bit integer value that is returned by the `GetDateTime` function. This is a long integer value that represents the number of seconds between midnight, January 1, 1904, and the time at which `GetDateTime` was called.

`longFlag`

A flag that indicates the desired format for the date string. This is one of the three values defined as the `DateForm` type.

The string produced by `DateString` is in one of three standard date formats used on the Macintosh, depending on which of the three `DateForm` values that you specify for the `longFlag` parameter: `shortDate`, `abbrevDate`, or `longDate`. The information in the supplied resource defines how month and day names are written and provides for calendars with more than 7 days and more than 12 months.

For the Roman script system's resource, the date January 31, 1992, produces the following three strings: "1/31/92", "Fri, Jan 31, 1992", and "Friday, January 31, 1992"(for `DateForm` values `shortDate`, `abbrevDate`, and `longDate`, respectively).

`result`

On output, contains the string representation of the date in the format indicated by the `longFlag` parameter.

intlHandle

A handle to a numeric-format or a long-date-format resource that specifies date formatting information for use in the conversion. If you specify `NULL` as the value of the resource handle parameter, `DateString` uses information from the current script. The numeric-format (`'itl0'`) resource specifies the short date formats and the long-date-format (`'itl1'`) resource specifies the long date formats.

`DateString` formats its data according to the information in the specified numeric-format resource (for short date formats) or long-date-format resource (for long date formats). If you specify `shortDate`, the `intlHandle` value should be the handle to a numeric-format resource; if you specify `abbrevDate` or `longDate`, it should be the handle to a long-date-format resource.

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
`DateTimeUtils.h`

DateToSeconds

Converts a date and time to a number of seconds elapsed since midnight, January 1, 1904. (Deprecated in Mac OS X v10.3. Use the `CFCalendarRef` data type and the functions that operate on it instead.)

```
void DateToSeconds (
   const DateTimeRec *d,
   unsigned long *secs
);
```

Parameters

d

The date-time structure containing the date and time to convert.

secs

On return, the number of seconds elapsed between midnight, January 1, 1904, and the time specified in the `d` parameter. For example, specifying a date and time of 11:33 A.M. on January 1, 1904 results in 41580 being returned in this parameter.

Special Considerations

For information on using the `CFCalendarRef` data type, see *Data Formatting Guide for Core Foundation* and *CFCalendar Reference*.

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
`DateTimeUtils.h`

GetDateTime

Obtains the current date-time information, expressed as the number of seconds elapsed since midnight, January 1, 1904. (Deprecated in Mac OS X v10.3. Use `CFAbsoluteTimeGetCurrent` instead.)

```
void GetDateTime (
   unsigned long *secs
);
```

Parameters

secs

On return, the number of seconds elapsed since midnight, January 1, 1904.

Discussion

The low-memory copy of the date and time information is also accessible through the global variable `Time`.

If an application disables interrupts for longer than a second, the date-time information returned by the `GetDateTime` function might not be exact. The `GetDateTime` function is intended to provide fairly accurate time information, but not scientifically precise data.

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

`DateTimeUtils.h`

GetLocalDateTime

Gets the local date and time. (Deprecated in Mac OS X v10.4. Use `CFAbsoluteTimeGetCurrent` and `CFTimeZoneGetSecondsFromGMT` instead.)

```
OSStatus GetLocalDateTime (
   LocalDateTime *localDateTime,
   OptionBits options
);
```

Parameters

localDateTime

A pointer to a value of type `LocalDateTime`. On return, the value this parameter points to is the current local date and time.

options

A value of type `OptionBits`. Pass `kUTCDefaultOptions` for the default behavior.

Return Value

A result code. See "Date, Time, and Measurement Utilities Result Codes" (page 421).

Discussion

This API returns the current date and time in `localTime`. Otherwise, it is set to 0. Use `kUTCDefaultOptions` in the options parameter for default behavior. Different behavior may be specified through this parameter in the future. If the operation is successful `noErr` is returned. If a `NULL` pointer is passed in the `localDateTime` parameter, `paramErr` is returned.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
UTCUtils.h

GetTime

Obtains the current date-time information, expressed as a date and time. (**Deprecated.** Use `CFAbsoluteTimeGetCurrent` instead.)

```
void GetTime (
   DateTimeRec *d
);
```

Parameters

d

On return, the fields of the date-time structure contain the current date and time.

Discussion
The `GetTime` function first calls the `GetDateTime` function to obtain the number of seconds elapsed since midnight, January 1, 1904. It then calls the `SecondsToDate` function to convert the number of seconds into a date and time.

As an alternative to using the `GetTime` procedure, you can pass the value of the global variable `Time` to the `SecondsToDate` (page 397) function; a `SecondsToDate(Time)` function call is identical to a `GetTime(d)` function call.

If an application disables interrupts for longer than a second, the date-time information returned by the `GetTime` function might not be exact. The `GetTime` function is intended to provide fairly accurate time information, but not scientifically precise data.

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

Declared In
QuickTimeComponents.k.h

GetUTCDateTime

Gets the UTC date and time. (Deprecated in Mac OS X v10.4. Use `CFAbsoluteTimeGetCurrent` instead.)

```
OSStatus GetUTCDateTime (
   UTCDateTime *utcDateTime,
   OptionBits options
);
```

Parameters

utcDateTime

A pointer to a value of type `UTCDateTime`. On return, the value this parameter points to is the current UTC date and time.

options

A value of type `OptionBits`. Pass `kUTCDefaultOptions` for the default behavior.

Return Value
A result code. See "Date, Time, and Measurement Utilities Result Codes" (page 421).

Discussion
This API returns the current date and time as UTC in `utcDateTime`. Otherwise, it is set to 0. Use `kUTCDefaultOptions` in the options for default behavior. Different behavior may be specified through this parameter in the future. If the operation is successful `noErr` is returned. If a `NULL` pointer is passed in `utcDateTime`, `paramErr` is returned.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`UTCUtils.h`

InitDateCache

Initializes the date cache structure, which is used to store data for use by the `StringToDate` and `StringToTime` functions. (Deprecated in Mac OS X v10.3. There is no replacement.)

```
OSErr InitDateCache (
   DateCachePtr theCache
);
```

Parameters

theCache
A pointer to a date cache structure. This parameter can be a local variable, a pointer, or a locked handle.

Return Value
A result code. See "Date, Time, and Measurement Utilities Result Codes" (page 421).

Discussion
You must call `InitDateCache` to initialize the date cache structure before using either the `StringToDate` (page 400) or `StringToTime` (page 401) functions. You must pass a pointer to a date cache structure. You have to declare the structure as a variable or allocate it in the heap.

If you are writing an application that allows the use of global variables, you can make your date cache structure a global variable and initialize it once, when you perform other global initialization.

`InitDateCache` calls the `GetResource` and `LoadResource` functions and it can also return the error codes they produce.

Special Considerations
You no longer need to initialize the data cache 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
`DateTimeUtils.h`

LongDateString

Converts a date that is specified as a `LongDateTime` value into a Pascal string, making use of the date formatting information in the specified resource. (Deprecated in Mac OS X v10.3. Use `CFDateFormatterCreateStringWithDate` instead.)

```
void LongDateString (
   const LongDateTime *dateTime,
   DateForm longFlag,
   Str255 result,
   Handle intlHandle
);
```

Parameters

dateTime

A pointer to a 64-bit, signed representation of the number of seconds since Jan. 1, 1904. This allows coverage of a much longer span of time (plus or minus approximately 30,000 years) than the standard, 32-bit representation.

longFlag

A flag that indicates the desired format for the date string. This is one of the three values defined as the `DateForm` type.

The string produced by `LongDateString` is in one of three standard date formats used on the Macintosh, depending on which of the three `DateForm` values that you specify for the `longFlag` parameter: `shortDate`, `abbrevDate`, or `longDate`. The information in the supplied resource defines how month and day names are written and provides for calendars with more than 7 days and more than 12 months.

For the U.S. resource, the date January 31, 1992, produces the following three strings: "1/31/92", "Fri, Jan 31, 1992", and "Friday, January 31, 1992"(for `DateForm` values `shortDate`, `abbrevDate`, and `longDate`, respectively).

result

On output, contains the string representation of the date in the format indicated by the `longFlag` parameter.

intlHandle

A handle to a numeric-format or long-date-format resource that specifies date formatting information for use in the conversion. If you specify `NULL` as the value of the resource handle parameter, `LongDateString` uses information from the current script. The numeric-format (`'itl0'`) resource specifies the short date formats and the long-date-format (`'itl1'`) resource specifies the long date formats.

If you specify `shortDate` in the `longFlag` parameter, the `intlHandle` value should be the handle to a numeric-format resource; if you specify `abbrevDate` or `longDate`, it should be the handle to a long-date-format resource.

Discussion

You can use the `LongSecondsToDate` and `LongDateToSeconds` functions to convert between the `LongDateRec` (as produced by the `StringToDate` function) and `LongDateTime` data types.

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

`DateTimeUtils.h`

LongDateToSeconds

Converts a date and time to the number of seconds elapsed since midnight, January 1, 1904. (Deprecated in Mac OS X v10.3. Use the `CFCalendarRef` data type and the functions that operate on it instead.)

```
void LongDateToSeconds (
   const LongDateRec *lDate,
   LongDateTime *lSecs
);
```

Parameters

lDate

The long date-time structure containing the date and time to convert.

lSecs

On return, the number of seconds elapsed since midnight, January 1, 1904, and the time specified in the `lDate` parameter. The number of seconds are returned as a long date-time value.

Special Considerations

For information on using the `CFCalendarRef` data type, see *Data Formatting Guide for Core Foundation* and *CFCalendar Reference*.

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

`DateTimeUtils.h`

LongSecondsToDate

Converts the number of seconds elapsed since midnight, January 1, 1904 to a date and time. (Deprecated in Mac OS X v10.3. Use the `CFCalendarRef` data type and the functions that operate on it instead.)

```
void LongSecondsToDate (
   const LongDateTime *lSecs,
   LongDateRec *lDate
);
```

Parameters

lSecs

The number of seconds elapsed since midnight, January 1, 1904.

lDate

On return, the fields of the long date-time structure that contain the date and time corresponding to the value indicated in the `lSecs` parameter. For example, specifying the number of seconds 41580 results in the date and time 11:33 A.M. on January 1, 1904 being returned in this parameter.

Special Considerations

For information on using the `CFCalendarRef` data type, see *Data Formatting Guide for Core Foundation* and *CFCalendar Reference*.

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
DateTimeUtils.h

LongTimeString

Converts a time that is specified as a LongDateTime value into a Pascal string, making use of the time formatting information in the specified resource. (Deprecated in Mac OS X v10.3. Use CFDateFormatterCreateStringWithDate instead.)

```
void LongTimeString (
   const LongDateTime *dateTime,
   Boolean wantSeconds,
   Str255 result,
   Handle intlHandle
);
```

Parameters

dateTime

A pointer to a 64-bit, signed representation of the number of seconds since Jan. 1, 1904. This allows coverage of a much longer span of time (plus or minus approximately 30,000 years) than the standard, 32-bit representation.

wantSeconds

A flag that indicates whether the seconds are to be included in the resulting string. LongTimeString produces a string that includes the seconds if you set this parameter to TRUE.

result

On output, contains the string representation of the time.

intlHandle

A handle to a numeric-format ('itl0') resource that specifies time formatting information for use in the conversion. If you specify NULL as the value of the resource handle parameter, LongTimeString uses information from the current script.

The numeric-format resource specifies whether or not to use leading zeros for the time values, whether to use a 12- or 24-hour time cycle, and how to specify morning or evening if a 12-hour time cycle is used.

Discussion
You can use the LongSecondsToDate and LongDateToSeconds functions to convert between the LongDateRec (as produced by the StringToTime (page 401) function) and LongDateTime data types.

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
DateTimeUtils.h

ReadDateTime

Reads time information from the system. (Deprecated in Mac OS X v10.3. Use `CFAbsoluteTimeGetCurrent` instead.)

```
OSErr ReadDateTime (
   unsigned long *time
);
```

Parameters

time

On return, the current time expressed as the number of seconds elapsed since midnight, January 1, 1904.

Return Value

A result code. See "Date, Time, and Measurement Utilities Result Codes" (page 421). If the clock chip cannot be read, `ReadDateTime` returns the `clkRdErr` result code. The operation might fail if the clock chip is damaged. Otherwise, the function returns the `noErr` result code.

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

`DateTimeUtils.h`

SecondsToDate

Converts a number of seconds elapsed since midnight, January 1, 1904 to a date and time. (Deprecated in Mac OS X v10.3. Use the `CFCalendarRef` data type and the functions that operate on it instead.)

```
void SecondsToDate (
   unsigned long secs,
   DateTimeRec *d
);
```

Parameters

secs

The number of seconds elapsed since midnight, January 1, 1904.

d

On return, the fields of the date-time structure that contain the date and time corresponding to the value indicated in the `s` parameter.

Special Considerations

For information on using the `CFCalendarRef` data type, see *Data Formatting Guide for Core Foundation* and *CFCalendar Reference*.

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
`DateTimeUtils.h`

SetDateTime

Changes the date-time information stored by the system to the specified value, expressed as the number of seconds elapsed since midnight, January 1, 1904. (Deprecated in Mac OS X v10.3. There is no replacement.)

```
OSErr SetDateTime (
   unsigned long time
);
```

Parameters

time

The number of seconds elapsed since midnight, January 1, 1904; this value is written to the system.

Return Value
A result code. See "Date, Time, and Measurement Utilities Result Codes" (page 421). The `SetDateTime` function attempts to verify the value written by reading it back in and comparing it to the value in the low-memory copy. If a problem occurs, the `SetDateTime` function returns either the `clkRdErr` result code, because the clock chip could not be read, or the `clkWrErr` result code, because the time written to the clock chip could not be verified. Otherwise, the function returns the `noErr` result code.

Special Considerations

Only the root user can set the time 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
`DateTimeUtils.h`

SetLocalDateTime

Sets the local date and time. (Deprecated in Mac OS X v10.4. There is no replacement.)

```
OSStatus SetLocalDateTime (
   const LocalDateTime *localDateTime,
   OptionBits options
);
```

Parameters

localDateTime

A pointer to a value of type `LocalDateTime` specifying the current local date and time.

options

A value of type `OptionBits`. Pass `kUTCDefaultOptions` for the default behavior.

Return Value
A result code. See "Date, Time, and Measurement Utilities Result Codes" (page 421).

Discussion
Use this call to set the clock to the date and time passed in the `localDateTime` parameter. Use `kUTCDefaultOptions` in the options for default behavior. Different behavior may be specified through this parameter in the future. If successful `noErr` is returned. Other errors include `kIllegalClockValueErr`, `paramErr` if `localDateTime` is `NULL`, or `clkWrErr` due to a failed attempt to write the value to the system.

Special Considerations

Only the root user can set the time 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
`UTCUtils.h`

SetTime

Changes the date-time information in the system to the specified value, expressed as a date and time. (Deprecated in Mac OS X v10.3. There is no replacement.)

```
void SetTime (
   const DateTimeRec *d
);
```

Parameters

d

The date and time to which to set in the system.

Discussion
The `SetTime` function first converts the date and time to the number of seconds elapsed since midnight, January 1, 1904 by calling the `DateToSeconds` function. It then writes these seconds to the system and to the system global variable `Time` by calling the `SetDateTime` function.

The `SetTime` function does not return a result code. If you need to know whether an attempt to change the date and time information in the system is successful, you must use the `SetDateTime` function.

As an alternative to using the `SetTime` procedure, you can use the `DateToSeconds` (page 390) and `SetDateTime` (page 398) functions.

Special Considerations

Only the root user can set the time 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
`DateTimeUtils.h`

SetUTCDateTime

Sets the UTC date and time. (Deprecated in Mac OS X v10.4. Use settimeofday(2) instead.)

```
OSStatus SetUTCDateTime (
   const UTCDateTime *utcDateTime,
   OptionBits options
);
```

Parameters

utcDateTime

A pointer to a value of type `UTCDateTime` specifying the current UTC date and time.

options

A value of type `OptionBits`. Pass `kUTCDefaultOptions` for the default behavior.

Return Value

A result code. See "Date, Time, and Measurement Utilities Result Codes" (page 421).

Discussion

Use this call to set the clock to the date and time passed in the `utcDateTime` parameter. Use `kUTCDefaultOptions` in the options for default behavior. Different behavior may be specified through this parameter in the future. If successful `noErr` is returned. Other errors include `kIllegalClockValueErr`, `kUTCUnderflowErr`, `kUTCOverflowErr`, and `paramErr` if `NULL` is passed for `utcDateTime`. It may also return `clkWrErr` due to a failed attempt to write the value to the system.

Special Considerations

Only the root user can set the time 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

`UTCUtils.h`

StringToDate

Parses a string for a date and converts the date information into values in a date-time structure. (Deprecated in Mac OS X v10.3. Use `CFDateFormatterCreateDateFromString` instead.)

```
StringToDateStatus StringToDate (
   Ptr textPtr,
   SInt32 textLen,
   DateCachePtr theCache,
   SInt32 *lengthUsed,
   LongDateRec *dateTime
);
```

Parameters

textPtr

A pointer to the text string to be parsed. `StringToDate` expects a date specification, in a format defined by the current script, at the beginning of the string.

textLen

The number of bytes in the text string.

theCache

A pointer to the date cache structure initialized by the `InitDateCache` (page 393) function with data that is used during the conversion process.

lengthUsed

On output, contains a pointer to the number of bytes of the string that were parsed for the date. Use this value to compute the starting location of the text that you can pass to `StringToTime` (page 401). Alternatively, you can use them in reverse order.

dateTime

On output, a pointer to the `LongDateRec` structure, which contains the year, month, day, and day of the week parsed for the date.

Return Value

A set of bit values that indicate confidence levels, with higher numbers indicating low confidence in how closely the input string matched what the function expected. For example, specifying a date with nonstandard separators may work, but it returns a message indicating that the separator was not standard. See the description of the `StringToDateStatus` data type.

Discussion

`StringToDate` parses the text string until it has finished finding all date information or until it has examined the number of bytes specified by `textLen`.

Note that `StringToDate` fills in only the year, month, day, and day of the week; `StringToTime` fills in the hour, minute, and second. You can use these two functions sequentially to fill in all of the values in a `LongDateRec` structure.

When one of the date components is missing, such as the year, the current date value is used as a default.

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

`DateTimeUtils.h`

StringToTime

Parses a string for a time specification and converts the date information into values in a date-time structure. (Deprecated in Mac OS X v10.3. Use `CFDateFormatterCreateDateFromString` instead.)

```
StringToDateStatus StringToTime (
   Ptr textPtr,
   SInt32 textLen,
   DateCachePtr theCache,
   SInt32 *lengthUsed,
   LongDateRec *dateTime
);
```

Parameters

textPtr

A pointer to the text string to be parsed. At the beginning of the string, `StringToTime` expects a time specification in a format defined by the current script.

textLen

The number of bytes in the text string.

theCache

A pointer to the date cache structure initialized by the `InitDateCache` function with data that is used during the conversion process.

lengthUsed

On output, contains a pointer to the length, in bytes, of the string that was parsed for the time.

dateTime

On output, a pointer to the `LongDateRec` structure, which contains the hour, minute, and second values that were parsed for the time.

Return Value

`StringToTime` returns a status value that indicates the confidence level for the success of the conversion. This is the same status value indicator type as does `StringToDate`: a set of bit values that indicate confidence levels, with higher numbers indicating low confidence in how closely the input string matched what the function expected. See the description of the `StringToDateStatus` data type.

Discussion

`StringToTime` parses the string until it has finished finding all time information or until it has examined the number of bytes specified by `textLen`.

Note that `StringToTime` fills in only the hour, minute, and second; `StringToDate` (page 400) fills in the year, month, day, and day of the week. You can use these two functions sequentially to fill in all of the values in a `LongDateRec` structure.

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

`DateTimeUtils.h`

TimeString

Converts a time in the standard date-time representation into a string, making use of the time formatting information in the specified resource. (Deprecated in Mac OS X v10.3. Use `CFDateFormatterCreateStringWithDate` instead.)

```
void TimeString (
   SInt32 dateTime,
   Boolean wantSeconds,
   Str255 result,
   Handle intlHandle
);
```

Parameters

dateTime

The date-time value in the representation returned by the Operating System function `GetDateTime`. The numeric representation used in these functions is the standard date-time representation: a 32-bit integer value that is returned by the `GetDateTime` function. This is a long integer value that represents the number of seconds between midnight, January 1, 1904, and the time at which `GetDateTime` was called.

wantSeconds

A flag that indicates whether the seconds are to be included in the resulting string.

result

On output, contains the string representation of the time.

intlHandle

A handle to a numeric-format (`'itl0'`) resource that specifies time formatting information for use in the conversion. If you specify `NULL` as the value of the resource handle parameter, `TimeString` uses information from the current script.

The numeric-format resource specifies whether or not to use leading zeros for the time values, whether to use a 12- or 24-hour time cycle, and how to specify morning or evening if a 12-hour time cycle is used.

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

`DateTimeUtils.h`

ToggleDate

Modifies a date and time, by modifying one specific component of a date and time (day, hour, minute, seconds, day of week, and so on). (Deprecated in Mac OS X v10.3. Use the `CFCalendarRef` data type and the functions that operate on it instead.)

```
ToggleResults ToggleDate (
   LongDateTime *lSecs,
   LongDateField field,
   DateDelta delta,
   short ch,
   const TogglePB *params
);
```

Parameters

lSecs

The date-time information to modify, expressed as the number of seconds elapsed since midnight, January 1, 1904.

field
The name of the field in the date-time structure you want modify. Use one of the Long Date Mask Constants (page 418) for the value of this parameter.

delta
A signed byte specifying the action you want to perform on the value specified in the `field` parameter. Set `delta` to 1, to increase the value in the field by 1. Set `delta` to -1, to decrease the value of the field by 1. Set `delta` to 0. If you want to set the value of the field explicitly; pass the new value through the `ch` field.

ch
If the value in the `delta` field is 0, the value of the field in the date-time structure (specified by the `field` parameter) is set to the value in the `ch` parameter. If the value in the `delta` field is not equal to 0, the value in the `ch` parameter is ignored.

params
The user-defined settings of the toggle parameter block settings.

Return Value
See the description of the `ToggleResults` data type.

Discussion
The relevant fields of the toggle parameter block are:

- `togFlags` A value of type `SInt32`. On input, the fields to be checked by the `ValidDate` function.
- `amChars` A value of type `ResType`. On input, A.M. characters from `'itl0'` resource.
- `pmChars` A value of type `ResType`. On input, P.M. characters from `'itl0'` resource.
- `reserved` An array of `SInt32` values. Reserved; on input, set each element to 0.

You must supply values for all input parameters.

The `ToggleDate` function first converts the number of seconds and makes each component of the date and time available through a long date-time structure. The `ToggleDate` function then modifies the value of the field, specified by the `field` parameter. If the value in the `delta` field is greater than 0, the value of the field increases by 1; if the value in the `delta` field is less than 0, the value of the field decreases by 1; and if the value of `delta` is 0, the value of the field is explicitly set to the value specified in the `ch` field. After the `ToggleDate` function modifies the field, it calls the `ValidDate` function. The `ValidDate` function checks the long date-time structure for correctness. If any of the structure fields are invalid, the `ValidDate` function returns a `LongDateField` value corresponding to the field in error. Otherwise, it returns the result code for `validDateFields`. Note that `ValidDate` reports only the least significant erroneous field.

After the `ToggleDate` function checks the validity of the modified field, it converts the modified date and time back into a number of seconds and returns these seconds in the `lSecs` parameter.

The `ToggleDate` function was previously available with the Script Manager.

For more information on the `LongDateRec` structure, see `LongDateRec` (page 413). The toggle parameter block structure is described in `TogglePB` (page 415).

For more information about the `GetIntlResource` function, see the Script Manager. For details on the `UppercaseText` function, see Text Utilities.

Special Considerations

For information on using the `CFCalendarRef` data type, see *Data Formatting Guide for Core Foundation* and *CFCalendar Reference*.

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
`DateTimeUtils.h`

UCConvertCFAbsoluteTimeToLongDateTime

Converts a value of type `CFAbsoluteTime` to `LongDateTime`.

```
OSStatus UCConvertCFAbsoluteTimeToLongDateTime (
   CFAbsoluteTime iCFTime,
   LongDateTime *oLongDate
);
```

Parameters

`iCFTime`
A CFAbsoluteTime value that represents the time from which you wish to convert.

`oLongDate`
A pointer to a value of type `LongDateTime`. On successful return, this will contain the converted time from the `CFAbsoluteTime` input.

Return Value
A result code. See "Date, Time, and Measurement Utilities Result Codes" (page 421).

Discussion
Use `UCConvertCFAbsoluteTimeToLongDateTime` to convert from a `CFAbsoluteTime` to a `LongDateTime`. Remember that the epoch for `LongDateTime` is January 1, 1904 while the epoch for `CFAbsoluteTime` is January 1, 2001.

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

Declared In
`DateTimeUtils.h`

UCConvertCFAbsoluteTimeToSeconds

Converts a value of type `CFAbsoluteTime` to seconds.

```
OSStatus UCConvertCFAbsoluteTimeToSeconds (
   CFAbsoluteTime iCFTime,
   UInt32 *oSeconds
);
```

Parameters

`iCFTime`
A `CFAbsoluteTime` value that represents the time from which you wish to convert.

oSeconds
A pointer to a value of type `UInt32`. On successful return, this contains the converted time from the `CFAbsoluteTime` input.

Return Value
A result code. See "Date, Time, and Measurement Utilities Result Codes" (page 421).

Discussion
Use `UCConvertCFAbsoluteTimeToSeconds` to convert from a `CFAbsoluteTime` to a *`UInt32`* representation of seconds. Remember that the epoch for seconds is January 1, 1904 while the epoch for CFAbsoluteTime is January 1, 2001.

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

Declared In
`DateTimeUtils.h`

UCConvertCFAbsoluteTimeToUTCDateTime

Converts a value of type `CFAbsoluteTime` to `UTCDateTime`.

```
OSStatus UCConvertCFAbsoluteTimeToUTCDateTime (
   CFAbsoluteTime iCFTime,
   UTCDateTime *oUTCDate
);
```

Parameters
`iCFTime`
A `CFAbsoluteTime` value that represents the time from which you wish to convert.

`oUTCDate`
A pointer to a `UTCDateTime`. On successful return, this will contain the converted time from the `CFAbsoluteTime` input.

Return Value
A result code. See "Date, Time, and Measurement Utilities Result Codes" (page 421).

Discussion
Use `UCConvertCFAbsoluteTimeToUTCDateTime` to convert from a `CFAbsoluteTime` to a `UTCDateTime`. Remember that the epoch for `UTCDateTime` is January 1, 1904 while the epoch for `CFAbsoluteTime` is January 1, 2001.

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

Declared In
`DateTimeUtils.h`

UCConvertLongDateTimeToCFAbsoluteTime

Converts a value of type `LongDateTime` to `CFAbsoluteTime`.

```
OSStatus UCConvertLongDateTimeToCFAbsoluteTime (
   LongDateTime iLongTime,
   CFAbsoluteTime *oCFTime
);
```

Parameters

iLongTime

A `LongDateTime` value that represents the time from which you wish to convert.

oCFTime

A pointer to a `CFAbsoluteTime`. On successful return, this will contain the converted time from the input time type.

Return Value
A result code. See "Date, Time, and Measurement Utilities Result Codes" (page 421).

Discussion
Use `UCConvertLongDateTimeToCFAbsoluteTime` to convert from a `LongDateTime` to a `CFAbsoluteTime`. Remember that the epoch for LongDateTime is January 1, 1904 while the epoch for `CFAbsoluteTime` is January 1, 2001.

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

Declared In
`DateTimeUtils.h`

UCConvertSecondsToCFAbsoluteTime

Converts a value from the normal seconds time representation to `CFAbsoluteTime`.

```
OSStatus UCConvertSecondsToCFAbsoluteTime (
   UInt32 iSeconds,
   CFAbsoluteTime *oCFTime
);
```

Parameters

iSeconds

A `UInt32` value that represents the time from which you wish to convert.

oCFTime

A pointer to a `CFAbsoluteTime`. On successful return, this will contain the converted time from the input time type.

Return Value
A result code. See "Date, Time, and Measurement Utilities Result Codes" (page 421).

Discussion
Use `UCConvertSecondsToCFAbsoluteTime` to convert from the normal seconds representation of time to a `CFAbsoluteTime`. Remember that the epoch for seconds is January 1, 1904 while the epoch for `CFAbsoluteTime` is January 1, 2001.

Keep in mind that this function converts local time (that is, the time in the local time zone) to GMT/UTC.

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

Declared In
`DateTimeUtils.h`

UCConvertUTCDateTimeToCFAbsoluteTime

Converts a value of type `UTCDateTime` time to `CFAbsoluteTime`.

```
OSStatus UCConvertUTCDateTimeToCFAbsoluteTime (
   const UTCDateTime *iUTCDate,
   CFAbsoluteTime *oCFTime
);
```

Parameters

iUTCDate
A pointer to a `UTCDateTime` structure that represents the time from which you wish to convert.

oCFTime
A pointer to a `CFAbsoluteTime`. On successful return, this contains the converted time from the input time type.

Return Value
A result code. See "Date, Time, and Measurement Utilities Result Codes" (page 421).

Discussion
Use `UCConvertUTCDateTimeToCFAbsoluteTime` to convert from a `UTCDDateTime` to a `CFAbsoluteTime`. Remember that the epoch for `UTCDateTime` is January 1, 1904 while the epoch for `CFAbsoluteTime` is January 1, 2001.

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

Declared In
`DateTimeUtils.h`

ValidDate

Verifies specific date and time values in a long date-time structure. (Deprecated in Mac OS X v10.3. Use the `CFCalendarRef` data type and the functions that operate on it instead.)

```
short ValidDate (
   const LongDateRec *vDate,
   long flags,
   LongDateTime *newSecs
);
```

Parameters

vDate
The long date-time structure whose fields you want to verify.

flags
The fields that you want to verify in the long date-time structure. For a description of the values you can use in this parameter, see Long Date Mask Constants (page 418).

newSecs
The date-time information, passed by the `ToggleDate` function, that you want to verify.

Return Value
If any of the specified fields contain invalid values, the `ValidDate` function returns a `LongDateField` value indicating the field in error. Otherwise, it returns the constant `validDateFields`. `ValidDate` reports only the least significant erroneous field.

Discussion
For more information on the `LongDateRec` structure, see `LongDateRec` (page 413). The toggle parameter block structure is described in `TogglePB` (page 415).

Special Considerations

For information on using the `CFCalendarRef` data type, see *Data Formatting Guide for Core Foundation* and *CFCalendar Reference*.

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
`DateTimeUtils.h`

Data Types

DateCacheRecord

```
struct DateCacheRecord {
    short hidden[256];
};
typedef struct DateCacheRecord DateCacheRecord;
typedef DateCacheRecord * DateCachePtr;
```

Fields
`hidden`
The storage used for converting dates and times.

Discussion
The `StringToDate` and `StringToTime` functions use the date cache, defined by the `DateCacheStructure` data type, as an area to store date conversion data that is used by the date conversion functions. This structure must be initialized by a call to the `InitDateCache` (page 393) function. The data in this structure is private—you should not attempt to access it.

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

Declared In
`DateTimeUtils.h`

DateDelta

```
typedef SInt8 DateDelta;
```

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

Declared In
`DateTimeUtils.h`

DateTimeRec

```
struct DateTimeRec {
    short year;
    short month;
    short day;
    short hour;
    short minute;
    short second;
    short dayOfWeek;
};
typedef struct DateTimeRec DateTimeRec;
```

Fields

`year`

The year, ranging from 1904 to 2040. Note that to indicate the year 1984, this field would store the integer 1984, not just 84. This field accepts input of 0 or negative values, but these values produce unpredictable results in the `year`, `month`, and `day` fields when you use the `SecondsToDate` and `DateToSeconds` functions. In addition, using `SecondsToDate` and `DateToSeconds` with year values greater than 2040 causes a wraparound to 1904 plus the number of years over 2040. For example, setting the year to 2045 returns a value of 1909, and the other fields in this record return unpredictable results.

`month`

The month of the year, where 1 represents January, and 12 represents December. Values greater than 12 cause a wraparound to a future year and month. This field accepts input of 0 or negative values, but these values produce unpredictable results in the `year`, `month`, and `day` fields when you use the `SecondsToDate` and `DateToSeconds` functions.

`day`

The day of the month, ranging from 1 to 31. Values greater than the number of days in a given month cause a wraparound to a future month and day. This feature is useful for working with leap years. For example, the 366th day of January in 1992 (1992 was a leap year) evaluates as December 31, 1992, and the 367th day of that year evaluates as January 1, 1993.

This field accepts 0 or negative values, but when you use the `SecondsToDate` and `DateToSeconds` procedures, a value of 0 in this field returns the last day of the previous month. For example, a month value of 2 and a day value of 0 return 1 and 31, respectively.

Using `SecondsToDate` and `DateToSeconds` with a negative number in this field subtracts that number of days from the last day in the previous month. For example, a month value of 5 and a day value of –1 return 4 for the month and 29 for the day a month value of 2 and a day value of –15 return 1 and 16, respectively.

`hour`

The hour of the day, ranging from 0 to 23, where 0 represents midnight and 23 represents 11:00 P.M. Values greater than 23 cause a wraparound to a future day and hour. This field accepts input of negative values, but these values produce unpredictable results in the `month`, `day`, `hour`, and `minute` fields you use the `SecondsToDate` and `DateToSeconds` procedures.

`minute`

The minute of the hour, ranging from 0 to 59. Values greater than 59 cause a wraparound to a future hour and minute. When you use the `SecondsToDate` and `DateToSeconds` procedures, a negative value in this field has the effect of subtracting that number from the beginning of the given hour. For example, an `hour` value of 1 and a `minute` value of –10 return 0 hours and 50 minutes. However, if the negative value causes the `hour` value to be less than 0, for example `hour` = 0, `minute` = –61, unpredictable results occur.

`second`

The second of the minute, ranging from 0 to 59. Values greater than 59 cause a wraparound to a future minute and second. When you use the `SecondsToDate` and `DateToSeconds` procedures, a negative value in this field has the effect of subtracting that number from the beginning of the given minute. For example, a `minute` value of 1 and a `second` value of –10 returns 0 minutes and 50 seconds. However, if the negative value causes the `hour` value to be less than 0, for example `hour` = 0, `minute` = 0, and `second` = –61, unpredictable results occur.

`dayOfWeek`

The day of the week, where 1 indicates Sunday and 7 indicates Saturday. This field accepts 0, negative values, or values greater than 7. When you use the `SecondsToDate` and `DateToSeconds` procedures, you get correct values because this field is automatically calculated from the values in the `year`, `month`, and `day` fields.

Discussion

The date-time record describes the date-time information as a date and time. The Date, Time, and Measurement Utilities use a date-time record to read and write date-time information to and from the system.

The date-time record can be used to hold date and time values only for a Gregorian calendar. The long date-time record, `LongDateRec` (page 413), can be used for a Gregorian calendar as well as other calendar systems.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DateTimeUtils.h`

LocalDateTime

```
struct LocalDateTime {
     UInt16 highSeconds;
     UInt32 lowSeconds;
     UInt16 fraction;
};
typedef struct LocalDateTime LocalDateTime;
typedef LocalDateTime * LocalDateTimePtr;
typedef LocalDateTimePtr * LocalDateTimeHandle;
```

Discussion
`UTCDateTime` and `LocalDateTime` are both 64 bits wide. The first 48 bits represent the number of seconds since 1904. The remaining 16 bits are used to indicate a fractional seconds value, which has no inherent precision. Each unit of this 16-bit value represents 1/65535 of a second. Developers may apply the appropriate arithmetic to derive milliseconds or microseconds.

Note that the decision to have the `lowSeconds` field divided between the high and low 32 bits of the 64 bit structure was intentional. The structure above is perfect for performing 64 bit math and logical comparisons. Having the `lowSeconds` field in the low or high 32 bits would have been easier for the compilers to handle and probably execute faster, however it would have rendered the structure unusable for 64 bit math and logical comparisons.

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

Declared In
`UTCUtils.h`

LongDateCvt

```
union LongDateCvt {
     SInt64 c
     struct {
         UInt32 lHigh;
         UInt32 lLow;
     } hl;
};
typedef union LongDateCvt LongDateCvt;
```

Fields

`c`
: The date and time, specified in seconds relative to midnight, January 1, 1904, as a signed, 64-bit integer in SANE `comp` format. The high-order bit of this field represents the sign of the 64-bit integer. Negative values allow you to indicate dates and times prior to midnight, January 1, 1904.

`hl`
: The high-order 32 bits when converting from a standard date-time value. Set this field to 0.

Discussion
The Date, Time, and Measurement Utilities provide the `LongDateCvt` structure to help in setting up `LongDateTime` values.

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

Declared In
`DateTimeUtils.h`

LongDateRec

```
union LongDateRec {
     struct {
         short era;
         short year;
         short month;
         short day;
         short hour;
         short minute;
         short second;
         short dayOfWeek;
         short dayOfYear;
         short weekOfYear;
         short pm;
         short res1;
         short res2;
         short res3;
     } ld;
     short list[14]
     struct {
         short eraAlt;
         DateTimeRec oldDate;
     } od;
};
typedef union LongDateRec LongDateRec;
```

Fields

`era`
The value `0` represents A.D. and `-1` represents B.C

`year`
The year, from 30081 B.C. to 29940 A.D.

`month`
The month (1 = January and 12 = December).

`day`
The day of the month, from 1 to 31.

`hour`
The hour, from 0 to 23.

`minute`
The minute, from 0 to 59.

`second`
The second., from 0 to 59

`dayOfWeek`
The day of the week (1 through 7).

`dayOfYear`
The day of the year, from 1 to 365.

`weekOfYear`
The week of the year. from 1 through 52.

`pm`
The value `0` represents AM and the value `1` represents PM.

`res1`
Reserved.

`res2`
Reserved.

`res3`
Reserved.

`list`
An array `[0 . . 13]` whose values indicate which of the fields in a long date-time record need to be verified.

`eraAlt`
Indicates the era, used only for conversion from a date-time record to a long date-time record.

`oldDate`
Used only for conversion from a date-time record to a long date-time record.

Discussion
In addition to the date-time record, system software provides the long date-time record, which extends the date-time record format by adding several more fields. This format lets you use dates and times with a much longer span (30,000 B.C. to 30,000 A.D.). In addition, the long date-time record allows conversions to different calendar systems, such as a lunar calendar.

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

Declared In
`DateTimeUtils.h`

LongDateTime

```
typedef SInt64 LongDateTime;
```

Discussion
The long date-time value specifies the date and time as seconds relative to midnight, January 1, 1904. But where the standard date-time value is an unsigned, 32-bit long integer, the long date-time value is a signed, 64-bit integer in SANE `comp` format. This format lets you use dates and times with a much longer span—roughly 500 billion years. You can use this value to represent dates and times prior to midnight, January 1, 1904. The `LongDateTime` data type defines the long date-time value.

When storing a long date-time value in files, you can use a 5-byte or 6-byte format for a range of roughly 35,000 years. You should sign extend this value to restore it to a `comp` format. Use the `LongDateCvt` (page 412) structure to help you in setting up a `LongDateTime` value.

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

Declared In
`DateTimeUtils.h`

String2DateStatus

```
typedef StringToDateStatus String2DateStatus;
```

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

Declared In
`DateTimeUtils.h`

StringToDateStatus

```
typedef short StringToDateStatus;
```

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

Declared In
`DateTimeUtils.h`

TogglePB

```
struct TogglePB {
     long togFlags;
     ResType amChars;
     ResType pmChars;
     long reserved[4];
};
typedef struct TogglePB TogglePB;
```

Fields

`togFlags`
: The high-order word of this field contains flags that specify special conditions for the `ToggleDate` function.

  The low-order word of this field contains masks representing fields to be checked by the `ValidDate` function. Each mask corresponds to a value in the enumerated type `LongDateField`. See Long Date Mask Constants (page 418) for a description of the values which you can use in this field. You can set this field to check the `era` through `second` fields by using the predeclared constant `dateStdMask`.

`amChars`
: The trailing string to display for morning (for example, A.M.). This string is read from the numeric-format resource (resource type `'itl0'`) of the current script system.

`pmChars`
: The trailing to display for evening (for example, P.M.). This string is read from the numeric-format resource (resource type `'itl0'`) of the current script system.

`reserved`
: Reserved. Set each of the three elements of this field to 0.

Discussion
The `ToggleDate` function exchanges information with your application using the toggle parameter block, defined by the `TogglePB` data type.

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

Declared In
`DateTimeUtils.h`

UTCDateTime

```
struct UTCDateTime {
     UInt16 highSeconds;
     UInt32 lowSeconds;
     UInt16 fraction;
};
typedef struct UTCDateTime UTCDateTime;
typedef UTCDateTime * UTCDateTimePtr;
typedef UTCDateTimePtr * UTCDateTimeHandle;
```

Discussion
`UTCDateTime` and `LocalDateTime` are both 64 bits wide. The first 48 bits represent the number of seconds since 1904. The remaining 16 bits are used to indicate a fractional seconds value, which has no inherent precision. Each unit of this 16-bit value represents 1/65535 of a second. Developers may apply the appropriate arithmetic to derive milliseconds or microseconds.

Note that the decision to divide the `lowSeconds` field between the high and low 32 bits of the 64 bit structure was intentional. You can use the structure to perform 64 bit math and logical comparisons. Having the `lowSeconds` field in the low or high 32 bits would have been easier for the compilers to handle and probably execute faster, however it would have rendered the structure unusable for 64 bit math and logical comparisons.

Important: You cannot access this structure as a `UInt64` data type. Doing so on systems that use little-endian byte odering may produce the wrong result.

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

Declared In
`UTCUtils.h`

Constants

Date Form Constants

```
typedef SInt8 DateForm;
enum {
     shortDate = 0,
     longDate = 1,
     abbrevDate = 2
};
```

Default Options

Options for use with the functions `SetDateTime` and `GetDateTime`.

```
enum {
     kUTCDefaultOptions = 0
};
```

Error Codes

```
enum {
     fatalDateTime = 0x8000,
     longDateFound = 1,
     leftOverChars = 2,
     sepNotIntlSep = 4,
     fieldOrderNotIntl = 8,
     extraneousStrings = 16,
     tooManySeps = 32,
     sepNotConsistent = 64,
     tokenErr = 0x8100,
     cantReadUtilities = 0x8200,
     dateTimeNotFound = 0x8400,
     dateTimeInvalid = 0x8800
};
```

Long Date Field Constants

```
typedef SInt8 LongDateField;
enum {
     eraField = 0,
     yearField = 1,
     monthField = 2,
     dayField = 3,
     hourField = 4,
     minuteField = 5,
     secondField = 6,
     dayOfWeekField = 7,
     dayOfYearField = 8,
     weekOfYearField = 9,
     pmField = 10,
     res1Field = 11,
     res2Field = 12,
     res3Field = 13
};
```

Long Date Mask Constants

```
enum {
     eraMask = 0x0001,
     yearMask = 0x0002,
     monthMask = 0x0004,
     dayMask = 0x0008,
     hourMask = 0x0010,
     minuteMask = 0x0020,
     secondMask = 0x0040,
     dayOfWeekMask = 0x0080,
     dayOfYearMask = 0x0100,
```

```
    weekOfYearMask = 0x0200,
    pmMask = 0x0400,
    dateStdMask = 0x007F
};
```

Constants

`eraMask`
Verify the era.
Available in Mac OS X v10.0 and later.
Declared in `DateTimeUtils.h`.

`yearMask`
Verify the year.
Available in Mac OS X v10.0 and later.
Declared in `DateTimeUtils.h`.

`monthMask`
Verify the month.
Available in Mac OS X v10.0 and later.
Declared in `DateTimeUtils.h`.

`dayMask`
Verify the day
Available in Mac OS X v10.0 and later.
Declared in `DateTimeUtils.h`.

`hourMask`
Verify the hour.
Available in Mac OS X v10.0 and later.
Declared in `DateTimeUtils.h`.

`minuteMask`
Verify the minute.
Available in Mac OS X v10.0 and later.
Declared in `DateTimeUtils.h`.

`secondMask`
Verify the second.
Available in Mac OS X v10.0 and later.
Declared in `DateTimeUtils.h`.

`dayOfWeekMask`
Verify the day of the week.
Available in Mac OS X v10.0 and later.
Declared in `DateTimeUtils.h`.

`dayOfYearMask`
Verify the day of the year.
Available in Mac OS X v10.0 and later.
Declared in `DateTimeUtils.h`.

`weekOfYearMask`
Verify the week of the year.

Available in Mac OS X v10.0 and later.

Declared in `DateTimeUtils.h`.

`pmMask`
Verify the evening (P.M.).

Available in Mac OS X v10.0 and later.

Declared in `DateTimeUtils.h`.

`dateStdMask`
Verify the era through the second.

Available in Mac OS X v10.0 and later.

Declared in `DateTimeUtils.h`.

Discussion
These constants are used in the `field` parameter of the `ToggleDate` (page 403) function to specify the `LongDateRec` fields for the `ValidDate` function to check.

Flags

```
enum {
     smallDateBit = 31,
     togChar12HourBit = 30,
     togCharZCycleBit = 29,
     togDelta12HourBit = 28,
     genCdevRangeBit = 27,
     validDateFields = -1,
     maxDateField = 10
};
```

Constants
`smallDateBit`
If this bit is set, the valid date and time are restricted to the range of the system global variable `Time`—that is, between midnight on January 1, 1904 and 6:28:15 A.M. on February 6, 2040.

Available in Mac OS X v10.0 and later.

Declared in `DateTimeUtils.h`.

`togChar12HourBit`
If this bit is set, modifying the hour by character is limited to the 12-hour range defined by `togCharZCycleBit`, mapped to the appropriate half of the 24-hour range, as determined by the `pm` field. This bit works with system software version 6.0.4 and later.

Available in Mac OS X v10.0 and later.

Declared in `DateTimeUtils.h`.

`togCharZCycleBit`
If this bit is set, the input character is treated as if it modifies an hour whose value is in the range 0–11. If this bit is not set, the input character is treated as if it modifies an hour whose value is in the range 12, 1–11. This bit works with system software version 6.0.4 and later.

Available in Mac OS X v10.0 and later.

Declared in `DateTimeUtils.h`.

`togDelta12HourBit`
: If this bit is set, modifying the hour up or down is limited to a 12-hour range. For example, increasing by one from 11 produces 0, increasing by one from 23 produces 12, and so on. This bit works with system software version 6.0.4 and later.

  Available in Mac OS X v10.0 and later.

  Declared in `DateTimeUtils.h`.

`genCdevRangeBit`
: If this bit is set in addition to `smallDateBit`, then the date range is restricted to that used by the General Controls control panel—January 1, 1920 to December 31, 2019 in the Gregorian calendar (the routine works correctly for other calendars as well). For dates outside this range but within the range specified by the system global variable `Time`—January 1, 1904 to February 6, 2040 in the Gregorian calendar—`ToggleDate` adds or subtracts 100 years to bring the dates into the range of the General Controls control panel if these bits are set. The `ToggleDate` function returns an error if the `smallDateBit` is set and the date is outside the range specified by the system global variable `Time`. This bit works with system software version 6.0.4 and later.

  Available in Mac OS X v10.0 and later.

  Declared in `DateTimeUtils.h`.

`validDateFields`
: Available in Mac OS X v10.0 and later.

  Declared in `DateTimeUtils.h`.

`maxDateField`
: Available in Mac OS X v10.0 and later.

  Declared in `DateTimeUtils.h`.

Toggle Results

```
typedef SInt16 ToggleResults;
enum {
     toggleUndefined = 0,
     toggleOK = 1,
     toggleBadField = 2,
     toggleBadDelta = 3,
     toggleBadChar = 4,
     toggleUnknown = 5,
     toggleBadNum = 6,
     toggleOutOfRange = 7,
     toggleErr3 = 7,
     toggleErr4 = 8,
     toggleErr5 = 9
};
```

Result Codes

The most common result codes returned by Date, Time, and Measurement Utilities are listed below.

Result Code	Value	Description
clkRdErr	-85	Unable to read the same clock value twice. Available in Mac OS X v10.0 and later.
clkWrErr	-86	The time written did not verify. Available in Mac OS X v10.0 and later.
kUTCUnderflowErr	-8850	An underflow error occurred. Available in Mac OS X v10.0 and later.
kUTCOverflowErr	-8851	An overflow error occurred. Available in Mac OS X v10.0 and later.
kIllegalClockValueErr	-8852	An illegal clock value was encountered. Available in Mac OS X v10.0 and later.

Debugger Services Reference

Framework: CoreServices/CoreServices.h
Declared in Debugging.h

Overview

Debugger Services is a Carbon API that provides standard exception handling and assertion functions to assist you in debugging Mac OS applications.

Functions by Task

Using Debugger Services

`NewDebugComponent` (page 429)
Registers a component with Debugger Services.

`NewDebugOption` (page 429)
Registers a new debug option with Debugger Services.

`GetDebugComponentInfo` (page 426)
Returns the signature and name of a registered component.

`GetDebugOptionInfo` (page 426)
Returns information about the debug option of a registered component.

`SetDebugOptionValue` (page 430)
Modifies the setting of a registered debug option.

`DisposeDebugComponent` (page 425)
Removes a component registration and all related debug options.

`DebugAssert` (page 424)
Displays an assertion messsage using the current output handler.

`InstallDebugAssertOutputHandler` (page 427)
Installs an output handler for `DebugAssert` to call in place of `DebugStr`, the default handler.

`TaskLevel` (page 431)
Provides information about the task interrupt level, if the task is running at interrupt-time.

Managing Callback UPPs

NewDebugAssertOutputHandlerUPP (page 428)

InvokeDebugAssertOutputHandlerUPP (page 428)

DisposeDebugAssertOutputHandlerUPP (page 425)

NewDebugComponentCallbackUPP (page 429)

InvokeDebugComponentCallbackUPP (page 428)

DisposeDebugComponentCallbackUPP (page 426)

Functions

DebugAssert

Displays an assertion messsage using the current output handler.

```
void DebugAssert (
   OSType componentSignature,
   UInt32 options,
   const char *assertionString,
   const char *exceptionLabelString,
   const char *errorString,
   const char *fileName,
   long lineNumber,
   void *value
);
```

Parameters

componentSignature
The unique signature of the component causing the assertion.

options
Reserved for use by Apple.

assertionString
A pointer to a string containing the assertion, or NULL.

exceptionLabelString
A pointer to a string containing the exceptionLabel, or NULL.

errorString
A pointer to the error string, or NULL.

fileName
A pointer to the file name or path name generated by the preprocessor __FILE__ identifier, or NULL.

lineNumber
The line number in the file (generated by the preprocessor __LINE__ identifier), or 0 (zero).

value
A value associated with the assertion, or NULL.

Discussion
The DEBUGASSERTMSG macro calls this function (by default) to display assertion messages. To redirect the output from this function, use InstallDebugAssertOutputHandler (page 427) to install a custom output handler.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X version 10.0 and later.

Declared In
Debugging.h

DisposeDebugAssertOutputHandlerUPP

```
void DisposeDebugAssertOutputHandlerUPP (
   DebugAssertOutputHandlerUPP userUPP
);
```

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X version 10.0 and later.

Declared In
Debugging.h

DisposeDebugComponent

Removes a component registration and all related debug options.

```
OSStatus DisposeDebugComponent (
   OSType componentSignature
);
```

Parameters

componentSignature
The unique signature of a component.

Return Value
A result code. If the result is non-zero, the Notification Manager cannot remove the debug options.

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

Declared In
Debugging.h

DisposeDebugComponentCallbackUPP

```
void DisposeDebugComponentCallbackUPP (
   DebugComponentCallbackUPP userUPP
);
```

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`Debugging.h`

GetDebugComponentInfo

Returns the signature and name of a registered component.

```
OSStatus GetDebugComponentInfo (
   UInt32 index,
   OSType *componentSignature,
   Str255 componentName
);
```

Parameters

index
An index into a list of registered components (one-based).

componentSignature
A pointer to an `OSType`, provided by the caller to receive the unique signature of the specified component.

componentName
A string buffer, provided by the caller to receive the component name.

Return Value
A result code. If `index` is not valid, the result code is `debuggingNoMatchErr`.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`Debugging.h`

GetDebugOptionInfo

Returns information about the debug option of a registered component.

```
OSStatus GetDebugOptionInfo (
   UInt32 index,
   OSType componentSignature,
   SInt32 *optionSelectorNum,
   Str255 optionName,
   Boolean *optionSetting
);
```

Parameters

index

An index into a list of registered debug options (zero-based). You should use the constant `kComponentDebugOption` (page 435).

componentSignature

The unique signature of your registered component.

optionSelectorNum

A pointer to an integer, provided by the caller to receive the option selector number.

optionName

A string buffer, provided by the caller to receive the option name.

optionSetting

A pointer to a `Boolean`, provided by the caller to receive the current option setting.

Return Value

A result code. Debugger Services returns `debuggingNoMatchErr` if the index is not valid, `debuggingInvalidSignatureErr` if the component is not registered, or `noErr`.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X version 10.0 and later.

Declared In

`Debugging.h`

InstallDebugAssertOutputHandler

Installs an output handler for `DebugAssert` to call in place of `DebugStr`, the default handler.

```
void InstallDebugAssertOutputHandler (
   DebugAssertOutputHandlerUPP handler
);
```

Parameters

handler

The custom output handler to install, or NULL to switch back to `DebugStr`.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X version 10.0 and later.

Declared In

`Debugging.h`

InvokeDebugAssertOutputHandlerUPP

```
void InvokeDebugAssertOutputHandlerUPP (
   OSType componentSignature,
   UInt32 options,
   const char *assertionString,
   const char *exceptionLabelString,
   const char *errorString,
   const char *fileName,
   long lineNumber,
   void *value,
   ConstStr255Param outputMsg,
   DebugAssertOutputHandlerUPP userUPP
);
```

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`Debugging.h`

InvokeDebugComponentCallbackUPP

```
void InvokeDebugComponentCallbackUPP (
   SInt32 optionSelectorNum,
   UInt32 command,
   Boolean *optionSetting,
   DebugComponentCallbackUPP userUPP
);
```

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`Debugging.h`

NewDebugAssertOutputHandlerUPP

```
DebugAssertOutputHandlerUPP NewDebugAssertOutputHandlerUPP (
   DebugAssertOutputHandlerProcPtr userRoutine
);
```

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`Debugging.h`

NewDebugComponent

Registers a component with Debugger Services.

```
OSStatus NewDebugComponent (
   OSType componentSignature,
   ConstStr255Param componentName,
   DebugComponentCallbackUPP componentCallback
);
```

Parameters

componentSignature

The unique signature of a new component.

componentName

A displayable string that names the new component.

componentCallback

A universal procedure pointer (UPP) to a debug component callback function, provided by the caller for working with options.

Return Value

A result code. See Debugger Services Result Codes (page 436).

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X version 10.0 and later.

Declared In

Debugging.h

NewDebugComponentCallbackUPP

```
DebugComponentCallbackUPP NewDebugComponentCallbackUPP (
   DebugComponentCallbackProcPtr userRoutine
);
```

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X version 10.0 and later.

Declared In

Debugging.h

NewDebugOption

Registers a new debug option with Debugger Services.

```
OSStatus NewDebugOption (
   OSType componentSignature,
   SInt32 optionSelectorNum,
   ConstStr255Param optionName
);
```

Parameters

componentSignature

The unique signature of a registered component.

optionSelectorNum

The selector number of the new debug option.

optionName

A displayable string that names this debug option.

Return Value

A result code. See `Debugger Services Result Codes` (page 436).

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X version 10.0 and later.

Declared In

`Debugging.h`

SetDebugOptionValue

Modifies the setting of a registered debug option.

```
OSStatus SetDebugOptionValue (
   OSType componentSignature,
   SInt32 optionSelectorNum,
   Boolean newOptionSetting
);
```

Parameters

componentSignature

The unique signature of a registered component.

optionSelectorNum

The selector number of a registered debug option.

newOptionSetting

The new setting for the option.

Return Value

A result code. Debugger Services returns `debuggingInvalidOptionErr` if the selector number is not valid, `debuggingInvalidSignatureErr` if the component is not registered, or `noErr`.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X version 10.0 and later.

Declared In

`Debugging.h`

TaskLevel

Provides information about the task interrupt level, if the task is running at interrupt-time.

```
UInt32 TaskLevel (
   void
);
```

Return Value
The current task interrupt level. If the return value is 0, the task is (probably) running at non-interrupt time. Otherwise, one of the `TaskLevel` masks can be used to learn more.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`Debugging.h`

Callbacks

DebugAssertOutputHandlerProcPtr

Defines a pointer to a function that handles the output from `DebugAssert` (page 424).

```
typedef void (*DebugAssertOutputHandlerProcPtr)
(
    OSType componentSignature,
    UInt32 options,
    const char * assertionString,
    const char * exceptionLabelString,
    const char * errorString,
    const char * fileName,
    long lineNumber,
    void * value,
    ConstStr255Param outputMsg
);
```

If you name your function `MyDebugAssertOutputHandler`, you would declare it like this:

```
void MyDebugAssertOutputHandler (
    OSType componentSignature,
    UInt32 options,
    const char * assertionString,
    const char * exceptionLabelString,
    const char * errorString,
    const char * fileName,
    long lineNumber,
    void * value,
    ConstStr255Param outputMsg
);
```

Parameters

componentSignature
The unique signature of the component causing the assertion.

options
Reserved for use by Apple.

assertionString
The name of the assertion, or NULL.

exceptionLabelString
The exception label, or NULL.

errorString
The description of an error condition, or NULL.

fileName
The file or path name (generated by the preprocessor __FILE__ identifier), or NULL.

fileName
The file or path name (generated by the preprocessor __FILE__ identifier), or NULL.

lineNumber
The line number in the file (generated by the preprocessor __LINE__ identifier), or 0 (zero).

value
A value associated with the assertion, or NULL.

outputMsg
The string that the caller (DebugAssert) normally passes to DebugStr when a custom output handler isn't installed.

Discussion
The parameters (excluding outputMsg) are the raw values passed to DebugAssert when an exception occurs. A custom output handler can safely ignore these parameters and simply redirect the output message (for example, to a log file).

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

Declared In
Debugging.h

DebugComponentCallbackProcPtr

Defines a pointer to a function that Debugger Services calls to read or modify the debug option settings defined by a component.

```
typedef void (*DebugComponentCallbackProcPtr)
(
    SInt32 optionSelectorNum,
    UInt32 command,
    Boolean * optionSetting
);
```

If you name your function MyDebugComponentCallback, you would declare it like this:

```
void MyDebugComponentCallback (
    SInt32 optionSelectorNum,
```

```
    UInt32 command,
    Boolean * optionSetting
);
```

Parameters

optionSelectorNum

A component debug option, previously defined by calling `NewDebugOption` (page 429).

command

Specifies the operation to be performed—`kGetDebugOption` to get current setting, or `kSetDebugOption` to modify the setting.

optionSetting

A pointer to a `Boolean` that Debugger Services uses to

- pass in the new setting, if `command` is `kSetDebugOption`
- receive the result of the operation, if `command` is `kGetDebugOption`

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

Declared In
`Debugging.h`

Data Types

DebugAssertOutputHandlerUPP

Defines a universal procedure pointer (UPP) type for a custom assertion output handler.

```
typedef DebugAssertOutputHandlerProcPtr DebugAssertOutputHandlerUPP;
```

Discussion
For information about custom assertion output handlers, see `DebugAssertOutputHandlerProcPtr` (page 431).

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

Declared In
`Debugging.h`

DebugComponentCallbackUPP

Defines a universal procedure pointer (UPP) type for a custom component debug option callback.

```
typedef DebugComponentCallbackProcPtr DebugComponentCallbackUPP;
```

Discussion
For information about custom component debug option callbacks, see `DebugComponentCallbackProcPtr` (page 432).

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

Declared In
`Debugging.h`

Constants

Interrupt Level Masks

Masks to determine what kind of tasks are executing at interrupt time.

```
enum {
    k68kInterruptLevelMask = 0x00000007,
    kInVBLTaskMask = 0x00000010,
    kInDeferredTaskMask = 0x00000020,
    kInSecondaryIntHandlerMask = 0x00000040,
    kInNestedInterruptMask = 0x00000080
};
```

Constants

`k68kInterruptLevelMask`
68K interrupt levels 0 through 7.
Available in Mac OS X v10.0 and later.
Declared in `Debugging.h`.

`kInVBLTaskMask`
VBLs are executing.
Available in Mac OS X v10.0 and later.
Declared in `Debugging.h`.

`kInDeferredTaskMask`
Deferred tasks are executing.
Available in Mac OS X v10.0 and later.
Declared in `Debugging.h`.

`kInSecondaryIntHandlerMask`
Secondary interrupt handlers are executing.
Available in Mac OS X v10.0 and later.
Declared in `Debugging.h`.

`kInNestedInterruptMask`
The operating system is handling an interrupt.
Available in Mac OS X v10.0 and later.
Declared in `Debugging.h`.

Discussion
For more information, see `TaskLevel` (page 431).

Unmapped Addresses

Addresses not mapped in Mac OS 8 or 9.

```
enum {
    kBlessedBusErrorBait = 0x68F168F1
};
```

Constants

`kBlessedBusErrorBait`

An address that will never be mapped in Mac OS 8 or 9.

Available in Mac OS X v10.0 and later.

Declared in `Debugging.h`.

Discussion

An exception occurs when an application tries to access the address `kBlessedBusErrorBait` in Mac OS 8 or 9, which makes it a good value to use when initializing pointers.

In Mac OS X, you should use 0x00000000 for this purpose.

Debug Option Types

Defines the debug option types supported by Debugger Services.

```
enum {
    kComponentDebugOption = 0
};
```

Constants

`kComponentDebugOption`

Specifies the component debug option type.

Available in Mac OS X v10.0 and later.

Declared in `Debugging.h`.

Discussion

For information about how this constant is used, see `GetDebugOptionInfo` (page 426).

Commands for Debug Option Callbacks

Defines the commands (or operations) that a debug option callback needs to implement.

```
enum {
    kGetDebugOption = 1,
    kSetDebugOption = 2
};
```

Constants

`kGetDebugOption`

The callback should return the current `Boolean` value of the specified debug option.

Available in Mac OS X v10.0 and later.

Declared in `Debugging.h`.

`kSetDebugOption`

The callback should modify the `Boolean` value of the specified debug option.

Available in Mac OS X v10.0 and later.

Declared in `Debugging.h`.

Result Codes

The most common result codes returned by Debugger Services are listed in the table below.

Result Code	Value	Description
`debuggingExecutionContextErr`	-13880	routine cannot be called at this time Available in Mac OS X v10.0 and later.
`debuggingDuplicateSignatureErr`	-13881	componentSignature already registered Available in Mac OS X v10.0 and later.
`debuggingDuplicateOptionErr`	-13882	optionSelectorNum already registered Available in Mac OS X v10.0 and later.
`debuggingInvalidSignatureErr`	-13883	componentSignature not registered Available in Mac OS X v10.0 and later.
`debuggingInvalidOptionErr`	-13884	optionSelectorNum is not registered Available in Mac OS X v10.0 and later.
`debuggingInvalidNameErr`	-13885	componentName or optionName is invalid (NULL) Available in Mac OS X v10.0 and later.
`debuggingNoCallbackErr`	-13886	debugging component has no callback Available in Mac OS X v10.0 and later.
`debuggingNoMatchErr`	-13887	debugging component or option not found at this index Available in Mac OS X v10.0 and later.

File Manager Reference

Framework: CoreServices/CoreServices.h

Declared in Files.h

Overview

The File Manager is a core service in Mac OS X that manages the organization, reading, and writing of data located on physical data storage devices such as disk drives. The File Manager provides an abstraction layer that hides lower-level implementation details such as different file systems and volume formats. If you want your application to have the same view of the file system seen in the Mac OS X user interface, the File Manager is an appropriate tool. For example, the File Manager is often used in application frameworks such as Carbon and Cocoa to implement file-related operations.

The File Manager API provides a large number of functions for performing various operations on files, directories, and volumes. The requirements of your application will dictate which of these functions you need to use. Many applications simply need to open files, read and write the data in those files, and then close the files. Other applications might provide more capabilities, such as the ability to copy or move a file to another directory. A few programs, such as the Mac OS X Finder, perform more extensive file operations and hence need to use some of the advanced functions provided by the File Manager.

A number of deprecated functions in the File Manager were inherited from earlier versions of Mac OS and have been carried along to facilitate porting legacy applications to Mac OS X. You should avoid using these deprecated functions. In particular, you should avoid any function or data structure that uses the `FSSpec` data type. This reference document clearly marks every deprecated function and, in most cases, provides a recommended replacement.

Functions by Task

Accessing Information About Files and Directories

`FSGetCatalogInfo` (page 494)
Returns catalog information about a file or directory. You can use this function to map an `FSRef` to an `FSSpec`.

`PBGetCatalogInfoSync` (page 647)
Returns catalog information about a file or directory. You can use this function to map from an `FSRef` to an `FSSpec`.

`PBGetCatalogInfoAsync` (page 643)
Returns catalog information about a file or directory. You can use this function to map from an `FSRef` to an `FSSpec`.

FSSetCatalogInfo (page 540)
Sets catalog information about a file or directory.

PBSetCatalogInfoSync (page 753)
Sets the catalog information about a file or directory.

PBSetCatalogInfoAsync (page 751)
Sets the catalog information about a file or directory.

FSpGetFInfo (page 530) Deprecated in Mac OS X v10.4
Obtains the Finder information for a file. (Deprecated. Use FSGetCatalogInfo (page 494) instead.)

FSpSetFInfo (page 535) Deprecated in Mac OS X v10.4
Sets the Finder information about a file. (Deprecated. Use FSSetCatalogInfo (page 540) instead.)

HGetFInfo (page 551) Deprecated in Mac OS X v10.4
Obtains the Finder information for a file. (Deprecated. Use FSGetCatalogInfo (page 494) instead.)

HSetFInfo (page 557) Deprecated in Mac OS X v10.4
Sets the Finder information for a file. (Deprecated. Use FSSetCatalogInfo (page 540) instead.)

PBGetCatInfoAsync (page 648) Deprecated in Mac OS X v10.4
Returns catalog information about a file or directory. (Deprecated. Use PBGetCatalogInfoAsync (page 643) instead.)

PBGetCatInfoSync (page 651) Deprecated in Mac OS X v10.4
Returns catalog information about a file or directory. (Deprecated. Use PBGetCatalogInfoSync (page 647) instead.)

PBHGetFInfoAsync (page 682) Deprecated in Mac OS X v10.4
Obtains information about a file. (Deprecated. Use PBGetCatalogInfoAsync (page 643) instead.)

PBHGetFInfoSync (page 683) Deprecated in Mac OS X v10.4
Obtains information about a file. (Deprecated. Use PBGetCatalogInfoSync (page 647) instead.)

PBHSetFInfoAsync (page 721) Deprecated in Mac OS X v10.4
Sets information for a file. (Deprecated. Use PBSetCatalogInfoAsync (page 751) instead.)

PBHSetFInfoSync (page 722) Deprecated in Mac OS X v10.4
Sets information for a file. (Deprecated. Use PBSetCatalogInfoSync (page 753) instead.)

PBSetCatInfoAsync (page 754) Deprecated in Mac OS X v10.4
Modifies catalog information for a file or directory. (Deprecated. Use PBSetCatalogInfoAsync (page 751) instead.)

PBSetCatInfoSync (page 755) Deprecated in Mac OS X v10.4
Modifies catalog information for a file or directory. (Deprecated. Use PBSetCatalogInfoSync (page 753) instead.)

Accessing the Desktop Database

PBDTAddAPPLAsync (page 602) Deprecated in Mac OS X v10.4
Adds an application to the desktop database. (Deprecated. There is no replacement function.)

PBDTAddAPPLSync (page 603) Deprecated in Mac OS X v10.4
Adds an application to the desktop database. (Deprecated. There is no replacement function.)

PBDTAddIconAsync (page 604) Deprecated in Mac OS X v10.4
Adds an icon definition to the desktop database. (Deprecated. There is no replacement function.)

PBDTAddIconSync (page 605) Deprecated in Mac OS X v10.4
Adds an icon definition to the desktop database. (Deprecated. There is no replacement function.)

PBDTCloseDown (page 606) Deprecated in Mac OS X v10.4
Closes the desktop database, though your application should never do this itself. (Deprecated. There is no replacement function.)

PBDTDeleteAsync (page 607) Deprecated in Mac OS X v10.4
Removes the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function. (Deprecated. There is no replacement function.)

PBDTDeleteSync (page 608) Deprecated in Mac OS X v10.4
Removes the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function. (Deprecated. There is no replacement function.)

PBDTFlushAsync (page 609) Deprecated in Mac OS X v10.4
Saves your changes to the desktop database. (Deprecated. There is no replacement function.)

PBDTFlushSync (page 610) Deprecated in Mac OS X v10.4
Saves your changes to the desktop database. (Deprecated. There is no replacement function.)

PBDTGetAPPLAsync (page 611) Deprecated in Mac OS X v10.4
Identifies the application that can open a file with a given creator. (Deprecated. There is no replacement function.)

PBDTGetAPPLSync (page 612) Deprecated in Mac OS X v10.4
Identifies the application that can open a file with a given creator. (Deprecated. There is no replacement function.)

PBDTGetCommentAsync (page 613) Deprecated in Mac OS X v10.4
Retrieves the user comments for a file or directory. (Deprecated. There is no replacement function.)

PBDTGetCommentSync (page 614) Deprecated in Mac OS X v10.4
Retrieves the user comments for a file or directory. (Deprecated. There is no replacement function.)

PBDTGetIconAsync (page 615) Deprecated in Mac OS X v10.4
Retrieves an icon definition. (Deprecated. There is no replacement function.)

PBDTGetIconInfoAsync (page 616) Deprecated in Mac OS X v10.4
Retrieves an icon type and the associated file type supported by a given creator in the desktop database. (Deprecated. There is no replacement function.)

PBDTGetIconInfoSync (page 618) Deprecated in Mac OS X v10.4
Retrieves an icon type and the associated file type supported by a given creator in the desktop database. (Deprecated. There is no replacement function.)

PBDTGetIconSync (page 619) Deprecated in Mac OS X v10.4
Retrieves an icon definition. (Deprecated. There is no replacement function.)

PBDTGetInfoAsync (page 620) Deprecated in Mac OS X v10.4
Determines information about the location and size of the desktop database on a particular volume. (Deprecated. There is no replacement function.)

PBDTGetInfoSync (page 621) Deprecated in Mac OS X v10.4
Determines information about the location and size of the desktop database on a particular volume. (Deprecated. There is no replacement function.)

PBDTGetPath (page 622) Deprecated in Mac OS X v10.4
Gets the reference number of the specified desktop database. (Deprecated. There is no replacement function.)

PBDTOpenInform (page 623) Deprecated in Mac OS X v10.4
Gets the reference number of the specified desktop database, reporting whether the desktop database was empty when it was opened. (Deprecated. There is no replacement function.)

PBDTRemoveAPPLAsync (page 624) Deprecated in Mac OS X v10.4
Removes an application from the desktop database. (Deprecated. There is no replacement function.)

PBDTRemoveAPPLSync (page 625) Deprecated in Mac OS X v10.4
Removes an application from the desktop database. (Deprecated. There is no replacement function.)

PBDTRemoveCommentAsync (page 626) Deprecated in Mac OS X v10.4
Removes a user comment associated with a file or directory from the desktop database. (Deprecated. There is no replacement function.)

PBDTRemoveCommentSync (page 627) Deprecated in Mac OS X v10.4
Removes a user comment associated with a file or directory from the desktop database. (Deprecated. There is no replacement function.)

PBDTResetAsync (page 627) Deprecated in Mac OS X v10.4
Removes information from the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function. (Deprecated. There is no replacement function.)

PBDTResetSync (page 628) Deprecated in Mac OS X v10.4
Removes information from the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function. (Deprecated. There is no replacement function.)

PBDTSetCommentAsync (page 629) Deprecated in Mac OS X v10.4
Adds a user comment for a file or a directory to the desktop database. (Deprecated. There is no replacement function.)

PBDTSetCommentSync (page 630) Deprecated in Mac OS X v10.4
Adds a user comment for a file or a directory to the desktop database. (Deprecated. There is no replacement function.)

Allocating Storage for Files

FSAllocateFork (page 470)
Allocates space on a volume to an open fork.

PBAllocateForkSync (page 567)
Allocates space on a volume to an open fork.

PBAllocateForkAsync (page 565)
Allocates space on a volume to an open fork.

Allocate (page 459) Deprecated in Mac OS X v10.4
Allocates additional space on a volume to an open file. (Deprecated. Use FSAllocateFork (page 470) instead.)

AllocContig (page 461) Deprecated in Mac OS X v10.4
Allocates additional contiguous space on a volume to an open file. (Deprecated. Use FSAllocateFork (page 470) instead.)

PBAllocateAsync (page 564) Deprecated in Mac OS X v10.4
Allocates additional space on a volume to an open file. (Deprecated. Use PBAllocateForkAsync (page 565) instead.)

`PBAllocateSync` (page 568) Deprecated in Mac OS X v10.4
: Allocates additional space on a volume to an open file. (Deprecated. Use `PBAllocateForkSync` (page 567) instead.)

`PBAllocContigAsync` (page 569) Deprecated in Mac OS X v10.4
: Allocates additional contiguous space on a volume to an open file. (Deprecated. Use `PBAllocateForkAsync` (page 565) instead.)

`PBAllocContigSync` (page 570) Deprecated in Mac OS X v10.4
: Allocates additional contiguous space on a volume to an open file. (Deprecated. Use `PBAllocateForkSync` (page 567) instead.)

Closing Files

`FSCloseFork` (page 475)
: Closes an open fork.

`PBCloseForkSync` (page 583)
: Closes an open fork.

`PBCloseForkAsync` (page 582)
: Closes an open fork.

`FSClose` (page 474)
: Closes an open file. (Deprecated. Use `FSCloseFork` (page 475) instead.)

`PBCloseAsync` (page 582) Deprecated in Mac OS X v10.5
: Closes an open file. (Deprecated. Use `PBCloseForkAsync` (page 582) instead.)

`PBCloseSync` (page 585) Deprecated in Mac OS X v10.5
: Closes an open file. (Deprecated. Use `PBCloseForkSync` (page 583) instead.)

Comparing File System References

`FSCompareFSRefs` (page 476)
: Determines whether two `FSRef` structures refer to the same file or directory.

`PBCompareFSRefsSync` (page 586)
: Determines whether two `FSRef` structures refer to the same file or directory.

`PBCompareFSRefsAsync` (page 586)
: Determines whether two `FSRef` structures refer to the same file or directory.

Controlling Directory Access

`PBHGetDirAccessAsync` (page 680) Deprecated in Mac OS X v10.5
: Returns the access control information for a directory or file. (Deprecated. Use `FSGetCatalogInfo` (page 494) instead.)

`PBHGetDirAccessSync` (page 681) Deprecated in Mac OS X v10.5
: Returns the access control information for a directory or file. (Deprecated. Use `FSGetCatalogInfo` (page 494) instead.)

PBHSetDirAccessAsync (page 719) Deprecated in Mac OS X v10.5
Changes the access control information for a directory. (Deprecated. Use FSSetCatalogInfo (page 540) instead.)

PBHSetDirAccessSync (page 720) Deprecated in Mac OS X v10.5
Changes the access control information for a directory. (Deprecated. Use FSSetCatalogInfo (page 540) instead.)

Controlling Login Access

PBHMapIDAsync (page 696) Deprecated in Mac OS X v10.5
Determines the name of a user or group given the user or group ID. (Deprecated. There is no replacement function.)

PBHMapIDSync (page 698) Deprecated in Mac OS X v10.5
Determines the name of a user or group given the user or group ID. (Deprecated. There is no replacement function.)

PBHMapNameAsync (page 698) Deprecated in Mac OS X v10.5
Determines the user ID or group ID from a user or group name. (Deprecated. There is no replacement function.)

PBHMapNameSync (page 700) Deprecated in Mac OS X v10.5
Determines the user ID or group ID from a user or group name. (Deprecated. There is no replacement function.)

PBHGetLogInInfoAsync (page 685) Deprecated in Mac OS X v10.4
Determines the login method used to log on to a particular shared volume. (Deprecated. There is no replacement function.)

Converting Between Paths and FSRef Structures

FSRefMakePath (page 539)
Converts an FSRef structure into a POSIX-style pathname.

FSPathMakeRef (page 519)
Converts a POSIX-style pathname into an FSRef structure.

FSPathMakeRefWithOptions (page 520)
Converts a POSIX-style pathname into an FSRef structure with options.

Copying and Moving Files

PBFSCopyFileSync (page 643)
Duplicates a file and optionally renames it.

PBFSCopyFileAsync (page 643)
Duplicates a file and optionally renames it.

PBHCopyFileAsync (page 673) Deprecated in Mac OS X v10.5
Duplicates a file and optionally renames it. (Deprecated. Use PBFSCopyFileAsync (page 643) instead.)

PBHCopyFileSync (page 675) Deprecated in Mac OS X v10.5
Duplicates a file and optionally renames it. (Deprecated. Use PBFSCopyFileSync (page 643) instead.)

PBHMoveRenameAsync (page 701) Deprecated in Mac OS X v10.4
Moves a file or directory and optionally renames it. (Deprecated. Use FSMoveObjectAsync (page 511) instead.)

PBHMoveRenameSync (page 702) Deprecated in Mac OS X v10.4
Moves a file or directory and optionally renames it. (Deprecated. Use FSMoveObjectSync (page 512) instead.)

Copying and Moving Objects Using Asynchronous High-Level File Operations

FSFileOperationCreate (page 488)
Creates an object that represents an asynchronous file operation.

FSFileOperationCancel (page 487)
Cancels an asynchronous file operation.

FSFileOperationGetTypeID (page 489)
Returns the Core Foundation type identifier for the FSFileOperation opaque type.

FSFileOperationScheduleWithRunLoop (page 489)
Schedules an asynchronous file operation with the specified run loop and mode.

FSFileOperationUnscheduleFromRunLoop (page 490)
Unschedules an asynchronous file operation from the specified run loop and mode.

FSCopyObjectAsync (page 477)
Starts an asynchronous file operation to copy a source object to a destination directory.

FSMoveObjectAsync (page 511)
Starts an asynchronous file operation to move a source object to a destination directory.

FSMoveObjectToTrashAsync (page 513)
Starts an asynchronous file operation to move a source object to the Trash.

FSPathCopyObjectAsync (page 517)
Starts an asynchronous file operation to copy a source object to a destination directory using pathnames.

FSPathMoveObjectAsync (page 521)
Starts an asynchronous file operation to move a source object to a destination directory using pathnames.

FSPathMoveObjectToTrashAsync (page 523)
Starts an asynchronous file operation to move a source object, specified using a pathname, to the Trash.

FSFileOperationCopyStatus (page 487)
Gets a copy of the current status information for an asynchronous file operation.

FSPathFileOperationCopyStatus (page 518)
Gets a copy of the current status information for an asynchronous file operation that uses pathnames.

Copying and Moving Objects Using Synchronous High-Level File Operations

FSCopyObjectSync (page 478)
Copies a source object to a destination directory.

FSMoveObjectSync (page 512)
Moves a source object to a destination directory.

FSMoveObjectToTrashSync (page 514)
Moves a source object to the Trash.

FSPathCopyObjectSync (page 518)
Copies a source object to a destination directory using pathnames.

FSPathMoveObjectSync (page 522)
Moves a source object to a destination directory using pathnames.

FSPathMoveObjectToTrashSync (page 524)
Moves a source object, specified using a pathname, to the Trash.

Creating a File System Reference (FSRef)

FSMakeFSRefUnicode (page 504)
Constructs an FSRef for a file or directory, given a parent directory and a Unicode name.

PBMakeFSRefUnicodeSync (page 733)
Constructs an FSRef for a file or directory, given a parent directory and a Unicode name.

PBMakeFSRefUnicodeAsync (page 733)
Constructs an FSRef for a file or directory, given a parent directory and a Unicode name.

FSpMakeFSRef (page 531) Deprecated in Mac OS X v10.5
Creates an FSRef for a file or directory, given an FSSpec. (Deprecated. There is no replacement function.)

PBMakeFSRefAsync (page 731) Deprecated in Mac OS X v10.5
Creates an FSRef for a file or directory, given an FSSpec. (Deprecated. Use PBMakeFSRefUnicodeAsync (page 733) instead.)

PBMakeFSRefSync (page 732) Deprecated in Mac OS X v10.5
Creates an FSRef for a file or directory, given an FSSpec. (Deprecated. Use PBMakeFSRefUnicodeSync (page 733) instead.)

Creating and Deleting File ID References

PBCreateFileIDRefAsync (page 590) Deprecated in Mac OS X v10.5
Establishes a file ID reference for a file. (Deprecated. Use FSGetCatalogInfo (page 494) instead.)

PBCreateFileIDRefSync (page 591) Deprecated in Mac OS X v10.5
Establishes a file ID reference for a file. (Deprecated. Use FSGetCatalogInfo (page 494) instead.)

PBDeleteFileIDRefAsync (page 596) Deprecated in Mac OS X v10.5
Deletes a file ID reference. (Deprecated. There is no replacement function.)

PBDeleteFileIDRefSync (page 597) Deprecated in Mac OS X v10.5
Deletes a file ID reference. (Deprecated. There is no replacement function.)

Creating and Deleting Named Forks

`FSCreateFork` (page 482)
Creates a named fork for a file or directory.

`PBCreateForkSync` (page 595)
Creates a named fork for a file or directory.

`PBCreateForkAsync` (page 594)
Creates a named fork for a file or directory.

`FSDeleteFork` (page 483)
Deletes a named fork from a file or directory.

`PBDeleteForkSync` (page 598)
Deletes a named fork from a file or directory.

`PBDeleteForkAsync` (page 597)
Deletes a named fork of a file or directory.

Creating Directories

`FSCreateDirectoryUnicode` (page 479)
Creates a new directory (folder) with a Unicode name.

`PBCreateDirectoryUnicodeSync` (page 589)
Creates a new directory (folder) with a Unicode name.

`PBCreateDirectoryUnicodeAsync` (page 587)
Creates a new directory (folder) with a Unicode name.

`DirCreate` (page 463) Deprecated in Mac OS X v10.4
Creates a new directory. (Deprecated. Use `FSCreateDirectoryUnicode` (page 479) instead.)

`FSpDirCreate` (page 527) Deprecated in Mac OS X v10.4
Creates a new directory. (Deprecated. Use `FSCreateDirectoryUnicode` (page 479) instead.)

`PBDirCreateAsync` (page 600) Deprecated in Mac OS X v10.4
Creates a new directory. (Deprecated. Use `PBCreateDirectoryUnicodeAsync` (page 587) instead.)

`PBDirCreateSync` (page 601) Deprecated in Mac OS X v10.4
Creates a new directory. (Deprecated. Use `PBCreateDirectoryUnicodeSync` (page 589) instead.)

Creating File System Specifications

`FSMakeFSSpec` (page 505) Deprecated in Mac OS X v10.4
Creates an `FSSpec` structure describing a file or directory. (Deprecated. Use `FSMakeFSRefUnicode` (page 504) instead.)

`PBMakeFSSpecAsync` (page 734) Deprecated in Mac OS X v10.4
Creates an `FSSpec` structure for a file or directory. (Deprecated. Use `PBMakeFSRefUnicodeAsync` (page 733) instead.)

`PBMakeFSSpecSync` (page 736) Deprecated in Mac OS X v10.4
Creates an `FSSpec` structure for a file or directory. (Deprecated. Use `PBMakeFSRefUnicodeSync` (page 733) instead.)

Creating Files

`FSCreateFileUnicode` (page 481)
Creates a new file with a Unicode name.

`PBCreateFileUnicodeSync` (page 593)
Creates a new file with a Unicode name.

`PBCreateFileUnicodeAsync` (page 591)
Creates a new file with a Unicode name.

`FSpCreate` (page 525) Deprecated in Mac OS X v10.4
Creates a new file. (Deprecated. Use `FSCreateFileUnicode` (page 481) instead.)

`HCreate` (page 550) Deprecated in Mac OS X v10.4
Creates a new file. (Deprecated. Use `FSCreateFileUnicode` (page 481) instead.)

`PBHCreateAsync` (page 676) Deprecated in Mac OS X v10.4
Creates a new file. (Deprecated. Use `PBCreateFileUnicodeAsync` (page 591) instead.)

`PBHCreateSync` (page 677) Deprecated in Mac OS X v10.4
Creates a new file. (Deprecated. Use `PBCreateFileUnicodeSync` (page 593) instead.)

Creating, Calling, and Deleting Universal Procedure Pointers

`NewIOCompletionUPP` (page 563)
Creates a new universal procedure pointer (UPP) to your I/O completion callback function.

`NewFNSubscriptionUPP` (page 562)
Creates a new universal procedure pointer (UPP) to your directory change callback function.

`NewFSVolumeEjectUPP` (page 562)
Creates a new universal procedure pointer (UPP) to your volume ejection callback function.

`NewFSVolumeMountUPP` (page 563)
Creates a new universal procedure pointer (UPP) to your volume mount callback function.

`NewFSVolumeUnmountUPP` (page 563)
Creates a new universal procedure pointer (UPP) to your volume unmount callback function.

`InvokeIOCompletionUPP` (page 561)
Calls your I/O completion callback function.

`InvokeFNSubscriptionUPP` (page 559)
Calls your directory change callback function.

`InvokeFSVolumeEjectUPP` (page 560)
Calls your volume ejection callback function.

`InvokeFSVolumeMountUPP` (page 560)
Calls your volume mount callback function.

`InvokeFSVolumeUnmountUPP` (page 561)
Calls your volume unmount callback function.

`DisposeIOCompletionUPP` (page 465)
Deletes a universal procedure pointer (UPP) to your I/O completion callback function.

`DisposeFNSubscriptionUPP` (page 464)
Deletes a universal procedure pointer (UPP) to your directory change callback function.

DisposeFSVolumeEjectUPP (page 464)
Deletes a universal procedure pointer (UPP) to your volume ejection callback function.

DisposeFSVolumeMountUPP (page 465)
Deletes a universal procedure pointer (UPP) to your volume mount callback function.

DisposeFSVolumeUnmountUPP (page 465)
Deletes a universal procedure pointer (UPP) to your volume unmount callback function.

Deleting Files and Directories

FSDeleteObject (page 484)
Deletes a file or an empty directory.

PBDeleteObjectSync (page 600)
Deletes a file or an empty directory.

PBDeleteObjectAsync (page 599)
Deletes a file or an empty directory.

FSpDelete (page 527) Deprecated in Mac OS X v10.4
Deletes a file or directory. (Deprecated. Use FSDeleteObject (page 484) instead.)

HDelete (page 551) Deprecated in Mac OS X v10.4
Deletes a file or directory. (Deprecated. Use FSDeleteObject (page 484) instead.)

PBHDeleteAsync (page 678) Deprecated in Mac OS X v10.4
Deletes a file or directory. (Deprecated. Use PBDeleteObjectAsync (page 599) instead.)

PBHDeleteSync (page 679) Deprecated in Mac OS X v10.4
Deletes a file or directory. (Deprecated. Use PBDeleteObjectSync (page 600) instead.)

Determining the Unicode Names of the Data and Resource Forks

FSGetDataForkName (page 497)
Returns a Unicode string constant for the name of the data fork.

FSGetResourceForkName (page 500)
Returns a Unicode string constant for the name of the resource fork.

Exchanging the Contents of Two Files

FSExchangeObjects (page 486)
Swaps the contents of two files.

PBExchangeObjectsSync (page 636)
Swaps the contents of two files.

PBExchangeObjectsAsync (page 635)
Swaps the contents of two files.

FSpExchangeFiles (page 528) Deprecated in Mac OS X v10.4
Exchanges the data stored in two files on the same volume. (Deprecated. Use FSExchangeObjects (page 486) instead.)

PBExchangeFilesAsync (page 631) Deprecated in Mac OS X v10.4
Exchanges the data stored in two files on the same volume. (Deprecated. Use PBExchangeObjectsAsync (page 635) instead.)

PBExchangeFilesSync (page 633) Deprecated in Mac OS X v10.4
Exchanges the data stored in two files on the same volume. (Deprecated. Use PBExchangeObjectsSync (page 636) instead.)

Getting and Setting Volume Information

FSGetVolumeInfo (page 500)
Returns information about a volume.

PBGetVolumeInfoSync (page 671)
Returns information about a volume.

PBGetVolumeInfoAsync (page 670)
Returns information about a volume.

FSSetVolumeInfo (page 543)
Sets information about a volume.

PBSetVolumeInfoSync (page 768)
Sets information about a volume.

PBSetVolumeInfoAsync (page 767)
Sets information about a volume.

FSCopyDiskIDForVolume (page 476)
Returns a copy of the disk ID for a volume.

FSCopyURLForVolume (page 479)
Returns a copy of the URL for a volume.

GetVRefNum (page 549) Deprecated in Mac OS X v10.4
Gets a volume reference number from a file reference number. (Deprecated. Use FSGetCatalogInfo (page 494) instead.)

PBHGetVInfoAsync (page 686) Deprecated in Mac OS X v10.4
Gets detailed information about a volume. (Deprecated. Use PBGetVolumeInfoAsync (page 670) instead.)

PBHGetVInfoSync (page 690) Deprecated in Mac OS X v10.4
Gets detailed information about a volume. (Deprecated. Use PBGetVolumeInfoSync (page 671) instead.)

PBSetVInfoAsync (page 765) Deprecated in Mac OS X v10.4
Changes information about a volume. (Deprecated. Use PBSetVolumeInfoAsync (page 767) instead.)

PBSetVInfoSync (page 766) Deprecated in Mac OS X v10.4
Changes information about a volume. (Deprecated. Use PBSetVolumeInfoSync (page 768) instead.)

PBXGetVolInfoAsync (page 779) Deprecated in Mac OS X v10.4
Returns information about a volume, including size information for volumes up to 2 terabytes. (Deprecated. Use FSGetVolumeInfo (page 500) instead.)

PBXGetVolInfoSync (page 782) Deprecated in Mac OS X v10.4
Returns information about a volume, including size information for volumes up to 2 terabytes. (Deprecated. Use FSGetVolumeInfo (page 500) instead.)

Getting Volume Attributes

`FSGetVolumeParms` (page 503)
Retrieves information about the characteristics of a volume.

`PBHGetVolParmsAsync` (page 694) Deprecated in Mac OS X v10.5
Returns information about the characteristics of a volume. (Deprecated. Use `FSGetVolumeParms` (page 503) instead.)

`PBHGetVolParmsSync` (page 695) Deprecated in Mac OS X v10.5
Returns information about the characteristics of a volume. (Deprecated. Use `FSGetVolumeParms` (page 503) instead.)

Iterating Over Named Forks

`FSIterateForks` (page 503)
Determines the name and size of every named fork belonging to a file or directory.

`PBIterateForksSync` (page 727)
Determines the name and size of every named fork belonging to a file or directory.

`PBIterateForksAsync` (page 726)
Determines the name and size of every named fork belonging to a file or directory.

Locking and Unlocking File Ranges

`FSLockRange` (page 504)
Locks a range of bytes of the specified fork.

`PBXLockRangeSync` (page 785)
Locks a range of bytes of the specified fork.

`PBXLockRangeAsync` (page 785)
Locks a range of bytes of the specified fork.

`FSUnlockRange` (page 543)
Unlocks a range of bytes of the specified fork.

`PBXUnlockRangeSync` (page 786)
Unlocks a range of bytes of the specified fork.

`PBXUnlockRangeAsync` (page 785)
Unlocks a range of bytes of the specified fork.

`PBLockRangeAsync` (page 728) Deprecated in Mac OS X v10.4
Locks a portion of a file. (Deprecated. Use `PBXLockRangeAsync` (page 785) instead.)

`PBLockRangeSync` (page 730) Deprecated in Mac OS X v10.4
Locks a portion of a file. (Deprecated. Use `PBXLockRangeSync` (page 785) or `FSLockRange` (page 504) instead.)

`PBUnlockRangeAsync` (page 770) Deprecated in Mac OS X v10.4
Unlocks a portion of a file. (Deprecated. Use `PBXUnlockRangeAsync` (page 785) instead.)

`PBUnlockRangeSync` (page 771) Deprecated in Mac OS X v10.4
Unlocks a portion of a file. (Deprecated. Use `PBXUnlockRangeSync` (page 786) or `FSUnlockRange` (page 543) instead.)

Locking and Unlocking Files and Directories

`FSpRstFLock` (page 534) Deprecated in Mac OS X v10.4
Unlocks a file or directory. (Deprecated. Use `FSSetCatalogInfo` (page 540) instead.)

`FSpSetFLock` (page 535) Deprecated in Mac OS X v10.4
Locks a file or directory. (Deprecated. Use `FSSetCatalogInfo` (page 540) instead.)

`HRstFLock` (page 556) Deprecated in Mac OS X v10.4
Unlocks a file or directory. (Deprecated. Use `FSSetCatalogInfo` (page 540) instead.)

`HSetFLock` (page 558) Deprecated in Mac OS X v10.4
Locks a file or directory. (Deprecated. Use `FSSetCatalogInfo` (page 540) instead.)

`PBHRstFLockAsync` (page 717) Deprecated in Mac OS X v10.4
Unlocks a file or directory. (Deprecated. Use `PBSetCatalogInfoAsync` (page 751) instead.)

`PBHRstFLockSync` (page 718) Deprecated in Mac OS X v10.4
Unlocks a file or directory. (Deprecated. Use `PBSetCatalogInfoSync` (page 753) instead.)

`PBHSetFLockAsync` (page 723) Deprecated in Mac OS X v10.4
Locks a file or directory. (Deprecated. Use `PBSetCatalogInfoAsync` (page 751) instead.)

`PBHSetFLockSync` (page 724) Deprecated in Mac OS X v10.4
Locks a file or directory. (Deprecated. Use `PBSetCatalogInfoSync` (page 753) instead.)

Manipulating File and Fork Size

`FSGetForkSize` (page 499)
Returns the size of an open fork.

`PBGetForkSizeSync` (page 665)
Returns the size of an open fork.

`PBGetForkSizeAsync` (page 664)
Returns the size of an open fork.

`FSSetForkSize` (page 542)
Changes the size of an open fork.

`PBSetForkSizeSync` (page 762)
Changes the size of an open fork.

`PBSetForkSizeAsync` (page 761)
Changes the size of an open fork.

`GetEOF` (page 548) Deprecated in Mac OS X v10.4
Determines the current logical size of an open file. (Deprecated. Use `FSGetForkSize` (page 499) instead.)

`PBGetEOFAsync` (page 654) Deprecated in Mac OS X v10.4
Determines the current logical size of an open file. (Deprecated. Use `PBGetForkSizeAsync` (page 664) instead.)

`PBGetEOFSync` (page 655) Deprecated in Mac OS X v10.4
Determines the current logical size of an open file. (Deprecated. Use `PBGetForkSizeSync` (page 665) instead.)

`PBSetEOFAsync` (page 757) Deprecated in Mac OS X v10.4
Sets the logical size of an open file. (Deprecated. Use `PBSetForkSizeAsync` (page 761) instead.)

PBSetEOFSync (page 758) Deprecated in Mac OS X v10.4
Sets the logical size of an open file. (Deprecated. Use PBSetForkSizeSync (page 762) instead.)

SetEOF (page 786) Deprecated in Mac OS X v10.4
Sets the logical size of an open file. (Deprecated. Use FSSetForkSize (page 542) instead.)

Manipulating File Position

FSGetForkPosition (page 499)
Returns the current position of an open fork.

PBGetForkPositionSync (page 663)
Returns the current position of an open fork.

PBGetForkPositionAsync (page 663)
Returns the current position of an open fork.

FSSetForkPosition (page 541)
Sets the current position of an open fork.

PBSetForkPositionSync (page 760)
Sets the current position of an open fork.

PBSetForkPositionAsync (page 759)
Sets the current position of an open fork.

GetFPos (page 548) Deprecated in Mac OS X v10.4
Returns the current position of the file mark. (Deprecated. Use FSGetForkPosition (page 499) instead.)

PBGetFPosAsync (page 666) Deprecated in Mac OS X v10.4
Returns the current position of the file mark. (Deprecated. Use PBGetForkPositionAsync (page 663) instead.)

PBGetFPosSync (page 666) Deprecated in Mac OS X v10.4
Returns the current position of the file mark. (Deprecated. Use PBGetForkPositionSync (page 663) instead.)

PBSetFPosAsync (page 763) Deprecated in Mac OS X v10.4
Sets the position of the file mark. (Deprecated. Use PBSetForkPositionAsync (page 759) instead.)

PBSetFPosSync (page 764) Deprecated in Mac OS X v10.4
Sets the position of the file mark. (Deprecated. Use PBSetForkPositionSync (page 760) instead.)

SetFPos (page 787) Deprecated in Mac OS X v10.4
Sets the position of the file mark. (Deprecated. Use FSSetForkPosition (page 541) instead.)

Manipulating the Default Volume

HGetVol (page 552) Deprecated in Mac OS X v10.4
Determines the current default volume and default directory. (Deprecated. There is no replacement function.)

HSetVol (page 559) Deprecated in Mac OS X v10.4
Sets the default volume and the default directory. (Deprecated. There is no replacement function.)

PBHGetVolAsync (page 693) Deprecated in Mac OS X v10.4
Determines the default volume and default directory. (Deprecated. There is no replacement function.)

PBHGetVolSync (page 695) Deprecated in Mac OS X v10.4
Determines the default volume and default directory. (Deprecated. There is no replacement function.)

PBHSetVolAsync (page 725) Deprecated in Mac OS X v10.4
Sets the default volume and the default directory. (Deprecated. There is no replacement function.)

PBHSetVolSync (page 726) Deprecated in Mac OS X v10.4
Sets the default volume and the default directory. (Deprecated. There is no replacement function.)

Mounting and Unmounting Volumes

FSMountLocalVolumeSync (page 507)
Mounts a volume.

FSMountServerVolumeSync (page 509)
Mounts a server volume.

FSUnmountVolumeSync (page 545)
Unmounts a volume.

FSEjectVolumeSync (page 486)
Ejects a volume.

FSCreateVolumeOperation (page 483)
Returns an FSVolumeOperation which can be used for an asynchronous volume operation.

FSCancelVolumeOperation (page 471)
Cancels an outstanding asynchronous volume mounting operation.

FSDisposeVolumeOperation (page 484)
Releases the memory associated with a volume operation.

FSMountLocalVolumeAsync (page 506)
Mounts a volume asynchronously.

FSMountServerVolumeAsync (page 508)
Mounts a server volume asynchronously.

FSUnmountVolumeAsync (page 544)
Unmounts a volume asynchronously.

FSEjectVolumeAsync (page 485)
Asynchronously ejects a volume.

FSGetAsyncMountStatus (page 492)
Returns the current status of an asynchronous mount operation.

FSGetAsyncUnmountStatus (page 493)
Returns the current status of an asynchronous unmount operation.

FSGetAsyncEjectStatus (page 491)
Returns the current status of an asynchronous eject operation.

PBUnmountVol (page 772) Deprecated in Mac OS X v10.4
Unmounts a volume. (Deprecated. Use FSEjectVolumeSync (page 486) or FSUnmountVolumeSync (page 545) instead.)

UnmountVol (page 787) Deprecated in Mac OS X v10.4
Unmounts a volume that isn't currently being used. (Deprecated. Use FSUnmountVolumeSync (page 545) instead.)

Mounting Remote Volumes

`FSGetVolumeMountInfoSize` (page 502)
Determines the size of the mounting information associated with the specified volume.

`FSGetVolumeMountInfo` (page 502)
Retrieves the mounting information associated with the specified volume.

`FSVolumeMount` (page 545)
Mounts a volume using the specified mounting information.

`PBGetVolMountInfo` (page 668) Deprecated in Mac OS X v10.5
Retrieves a record containing all the information needed to mount a volume, except for passwords. (Deprecated. Use `FSVolumeMount` (page 545) instead.)

`PBGetVolMountInfoSize` (page 669) Deprecated in Mac OS X v10.5
Determines how much space to allocate for a volume mounting information structure. (Deprecated. Use `FSVolumeMount` (page 545) instead.)

`PBVolumeMount` (page 773) Deprecated in Mac OS X v10.5
Mounts a volume. (Deprecated. Use `FSVolumeMount` (page 545) instead.)

Moving and Renaming Files or Directories

`FSMoveObject` (page 510)
Moves a file or directory into a different directory.

`PBMoveObjectSync` (page 738)
Moves a file or directory into a different directory.

`PBMoveObjectAsync` (page 737)
Moves a file or directory into a different directory.

`FSRenameUnicode` (page 539)
Renames a file or folder.

`PBRenameUnicodeSync` (page 748)
Renames a file or folder.

`PBRenameUnicodeAsync` (page 748)
Renames a file or folder.

`CatMove` (page 462) Deprecated in Mac OS X v10.4
Moves files or directories from one directory to another on the same volume. (Deprecated. Use `FSMoveObject` (page 510) instead.)

`FSpCatMove` (page 524) Deprecated in Mac OS X v10.4
Moves a file or directory from one location to another on the same volume. (Deprecated. Use `FSMoveObject` (page 510) instead.)

`FSpRename` (page 533) Deprecated in Mac OS X v10.4
Renames a file or directory. (Deprecated. Use `FSRenameUnicode` (page 539) instead.)

`HRename` (page 555) Deprecated in Mac OS X v10.4
Renames a file, directory, or volume. (Deprecated. Use `FSRenameUnicode` (page 539) instead.)

`PBCatMoveAsync` (page 575) Deprecated in Mac OS X v10.4
Moves files or directories from one directory to another on the same volume. (Deprecated. Use `PBMoveObjectAsync` (page 737) instead.)

`PBCatMoveSync` (page 576) Deprecated in Mac OS X v10.4
Moves files or directories from one directory to another on the same volume. (Deprecated. Use `PBMoveObjectSync` (page 738) instead.)

`PBHRenameAsync` (page 715) Deprecated in Mac OS X v10.4
Renames a file, directory, or volume. (Deprecated. Use `PBRenameUnicodeAsync` (page 748) instead.)

`PBHRenameSync` (page 716) Deprecated in Mac OS X v10.4
Renames a file, directory, or volume. (Deprecated. Use `PBRenameUnicodeSync` (page 748) instead.)

Obtaining File and Directory Information Using a Catalog Iterator on HFS Plus Volumes

`FSGetCatalogInfoBulk` (page 495)
Returns information about one or more objects from a catalog iterator. This function can return information about multiple objects in a single call.

`PBGetCatalogInfoBulkSync` (page 646)
Returns information about one or more objects from a catalog iterator. This function can return information about multiple objects in a single call.

`PBGetCatalogInfoBulkAsync` (page 644)
Returns information about one or more objects from a catalog iterator. This function can return information about multiple objects in a single call.

Obtaining File Control Block Information

`PBGetFCBInfoAsync` (page 656) Deprecated in Mac OS X v10.4
Gets information about an open file from the file control block. (Deprecated. Use `PBGetForkCBInfoAsync` (page 660) instead.)

`PBGetFCBInfoSync` (page 658) Deprecated in Mac OS X v10.4
Gets information about an open file from the file control block. (Deprecated. Use `PBGetForkCBInfoSync` (page 661) instead.)

Obtaining Fork Control Block Information

`FSGetForkCBInfo` (page 497)
Returns information about a specified open fork, or about all open forks.

`PBGetForkCBInfoSync` (page 661)
Returns information about a specified open fork, or about all open forks.

`PBGetForkCBInfoAsync` (page 660)
Returns information about a specified open fork, or about all open forks.

Opening Files

`FSOpenFork` (page 514)
Opens any fork of a file or directory for streaming access.

PBOpenForkSync (page 740)
Opens any fork of a file or directory for streaming access.

PBOpenForkAsync (page 739)
Opens any fork of a file or directory for streaming access.

FSpOpenDF (page 531) Deprecated in Mac OS X v10.4
Opens the data fork of a file. (Deprecated. Use FSOpenFork (page 514) instead.)

FSpOpenRF (page 532) Deprecated in Mac OS X v10.4
Opens the resource fork of a file. (Deprecated. Use FSOpenFork (page 514) instead.)

HOpen (page 553) Deprecated in Mac OS X v10.4
Opens the data fork of a file. (Deprecated. Use FSOpenFork (page 514) instead.)

HOpenDF (page 554) Deprecated in Mac OS X v10.4
Opens the data fork of a file. (Deprecated. Use FSOpenFork (page 514) instead.)

HOpenRF (page 554) Deprecated in Mac OS X v10.4
Opens the resource fork of a file. (Deprecated. Use FSOpenFork (page 514) instead.)

PBHOpenAsync (page 703) Deprecated in Mac OS X v10.4
Opens the data fork of a file. (Deprecated. Use PBOpenForkAsync (page 739) instead.)

PBHOpenDFAsync (page 706) Deprecated in Mac OS X v10.4
Opens the data fork of a file. (Deprecated. Use PBOpenForkAsync (page 739) instead.)

PBHOpenDFSync (page 708) Deprecated in Mac OS X v10.4
Opens the data fork of a file. (Deprecated. Use PBOpenForkSync (page 740) instead.)

PBHOpenRFAsync (page 709) Deprecated in Mac OS X v10.4
Opens the resource fork of a file. (Deprecated. Use PBOpenForkAsync (page 739) instead.)

PBHOpenRFSync (page 713) Deprecated in Mac OS X v10.4
Opens the resource fork of a file. (Deprecated. Use PBOpenForkSync (page 740) instead.)

PBHOpenSync (page 714) Deprecated in Mac OS X v10.4
Opens the data fork of a file. (Deprecated. Use PBOpenForkSync (page 740) instead.)

Opening Files While Denying Access

PBHOpenDenyAsync (page 704) Deprecated in Mac OS X v10.5
Opens a file's data fork using the access deny modes. (Deprecated. Use PBOpenForkAsync (page 739) with deny modes in the permissions field.)

PBHOpenDenySync (page 705) Deprecated in Mac OS X v10.5
Opens a file's data fork using the access deny modes. (Deprecated. Use PBOpenForkSync (page 740) with deny modes in the permissions field.)

PBHOpenRFDenyAsync (page 710) Deprecated in Mac OS X v10.5
Opens a file's resource fork using the access deny modes. (Deprecated. Use PBOpenForkAsync (page 739) with deny modes in the permissions field.)

PBHOpenRFDenySync (page 711) Deprecated in Mac OS X v10.5
Opens a file's resource fork using the access deny modes. (Deprecated. Use PBOpenForkSync (page 740) with deny modes in the permissions field.)

Reading and Writing Files

`FSReadFork` (page 537)
Reads data from an open fork.

`PBReadForkSync` (page 745)
Reads data from an open fork.

`PBReadForkAsync` (page 744)
Reads data from an open fork.

`FSWriteFork` (page 546)
Writes data to an open fork.

`PBWriteForkSync` (page 777)
Writes data to an open fork.

`PBWriteForkAsync` (page 776)
Writes data to an open fork.

`PBReadAsync` (page 743) Deprecated in Mac OS X v10.5
Reads any number of bytes from an open file. (Deprecated. Use `PBReadForkAsync` (page 744) instead.)

`PBReadSync` (page 746) Deprecated in Mac OS X v10.5
Reads any number of bytes from an open file. (Deprecated. Use `PBReadForkSync` (page 745) instead.)

`PBWriteAsync` (page 775) Deprecated in Mac OS X v10.5
Writes any number of bytes to an open file. (Deprecated. Use `PBWriteForkAsync` (page 776) instead.)

`PBWriteSync` (page 779) Deprecated in Mac OS X v10.5
Writes any number of bytes to an open file. (Deprecated. Use `PBWriteForkSync` (page 777) instead.)

`FSRead` (page 536) Deprecated in Mac OS X v10.4
Reads any number of bytes from an open file. (Deprecated. Use `FSReadFork` (page 537) instead.)

`FSWrite` (page 546) Deprecated in Mac OS X v10.4
Writes any number of bytes to an open file. (Deprecated. Use `FSWriteFork` (page 546) instead.)

Resolving File ID References

`PBResolveFileIDRefAsync` (page 749) Deprecated in Mac OS X v10.5
Retrieves the filename and parent directory ID of a file given its file ID. (Deprecated. Use `FSGetCatalogInfo` (page 494) instead.)

`PBResolveFileIDRefSync` (page 750) Deprecated in Mac OS X v10.5
Retrieves the filename and parent directory ID of a file given its file ID. (Deprecated. Use `FSGetCatalogInfo` (page 494) instead.)

Searching a Volume

`PBCatSearchAsync` (page 577) Deprecated in Mac OS X v10.4
Searches a volume's catalog file using a set of search criteria that you specify. (Deprecated. Use `PBCatalogSearchAsync` (page 572) instead.)

`PBCatSearchSync` (page 580) Deprecated in Mac OS X v10.4
Searches a volume's catalog file using a set of search criteria that you specify. (Deprecated. Use `PBCatalogSearchSync` (page 573) instead.)

Searching a Volume Using a Catalog Iterator

`FSOpenIterator` (page 515)
Creates a catalog iterator that can be used to iterate over the contents of a directory or volume.

`PBOpenIteratorSync` (page 742)
Creates a catalog iterator that can be used to iterate over the contents of a directory or volume.

`PBOpenIteratorAsync` (page 741)
Creates a catalog iterator that can be used to iterate over the contents of a directory or volume.

`FSCatalogSearch` (page 472)
Searches for objects traversed by a catalog iterator that match a given set of criteria.

`PBCatalogSearchSync` (page 573)
Searches for objects traversed by a catalog iterator that match a given set of criteria.

`PBCatalogSearchAsync` (page 572)
Searches for objects traversed by a catalog iterator that match a given set of criteria.

`FSCloseIterator` (page 475)
Closes a catalog iterator.

`PBCloseIteratorSync` (page 584)
Closes a catalog iterator.

`PBCloseIteratorAsync` (page 584)
Closes a catalog iterator.

Updating Files

`FSFlushFork` (page 490)
Causes all data written to an open fork to be written to disk.

`PBFlushForkSync` (page 639)
Causes all data written to an open fork to be written to disk.

`PBFlushForkAsync` (page 638)
Causes all data written to an open fork to be written to disk.

`PBFlushFileAsync` (page 636) Deprecated in Mac OS X v10.4
Writes the contents of a file's access path buffer to the disk. (Deprecated. Use `PBFlushForkAsync` (page 638) instead.)

`PBFlushFileSync` (page 637) Deprecated in Mac OS X v10.4
Writes the contents of a file's access path buffer to the disk. (Deprecated. Use `PBFlushForkSync` (page 639) instead.)

Updating Volumes

`FSFlushVolume` (page 491)
For the specified volume, writes all open and modified files in the current process to permanent storage.

`PBFlushVolumeSync` (page 642)
For the specified volume, writes all open and modified files in the current process to permanent storage.

`PBFlushVolumeAsync` (page 642)
For the specified volume, writes all open and modified files in the current process to permanent storage.

`FlushVol` (page 466) Deprecated in Mac OS X v10.5
Writes the contents of the volume buffer and updates information about the volume. (Deprecated. Use `FSFlushVolume` (page 491) instead.)

`PBFlushVolAsync` (page 640) Deprecated in Mac OS X v10.5
Writes the contents of the volume buffer and updates information about the volume. (Deprecated. Use `PBFlushVolumeAsync` (page 642) instead.)

`PBFlushVolSync` (page 641) Deprecated in Mac OS X v10.5
Writes the contents of the volume buffer and updates information about the volume. (Deprecated. Use `PBFlushVolumeSync` (page 642) instead.)

Using Change Notifications

`FNNotify` (page 467)
Broadcasts notification of changes to the specified directory.

`FNNotifyAll` (page 468)
Broadcasts notification of changes to the filesystem.

`FNNotifyByPath` (page 468)
Broadcasts notification of changes to the specified directory.

`FNSubscribe` (page 469)
Subscribes to change notifications for the specified directory.

`FNSubscribeByPath` (page 469)
Subscribes to change notifications for the specified directory.

`FNUnsubscribe` (page 470)
Releases a subscription which is no longer needed.

`FNGetDirectoryForSubscription` (page 466)
Fetches the directory for which this subscription was originally entered.

Not Recommended

This section lists functions that are not recommended and you should no longer use.

`PBWaitIOComplete` (page 775) Deprecated in Mac OS X v10.5
Keeps the system idle until either an interrupt occurs or the specified timeout value is reached. (Deprecated. There is no replacement function.)

`PBGetForeignPrivsAsync` (page 659) Deprecated in Mac OS X v10.4
Determines the native access-control information for a file or directory stored on a volume managed by a foreign file system. (Deprecated. There is no replacement function.)

`PBGetForeignPrivsSync` (page 660) Deprecated in Mac OS X v10.4
Determines the native access-control information for a file or directory stored on a volume managed by a foreign file system. (Deprecated. There is no replacement function.)

`PBGetUGEntryAsync` (page 667) Deprecated in Mac OS X v10.4
Gets a user or group entry from the list of User and Group names and IDs on the local file server. (Deprecated. There is no replacement function.)

PBGetUGEntrySync (page 668) Deprecated in Mac OS X v10.4
: Gets a user or group entry from the list of User and Group names and IDs on a local file server. (Deprecated. There is no replacement function.)

PBGetXCatInfoAsync (page 672) Deprecated in Mac OS X v10.4
: Returns the short name (MS-DOS format name) and the ProDOS information for a file or directory. (Deprecated. There is no replacement function.)

PBGetXCatInfoSync (page 673) Deprecated in Mac OS X v10.4
: Returns the short name (MS-DOS format name) and the ProDOS information for a file or directory. (Deprecated. There is no replacement function.)

PBHGetLogInInfoSync (page 686) Deprecated in Mac OS X v10.4
: Determines the login method used to log on to a particular shared volume. (Deprecated. There is no replacement function.)

PBSetForeignPrivsAsync (page 759) Deprecated in Mac OS X v10.4
: Changes the native access-control information for a file or directory stored on a volume managed by a foreign file system. (Deprecated. There is no replacement function.)

PBSetForeignPrivsSync (page 759) Deprecated in Mac OS X v10.4
: Changes the native access-control information for a file or directory stored on a volume managed by a foreign file system. (Deprecated. There is no replacement function.)

PBShareAsync (page 769) Deprecated in Mac OS X v10.4
: Establishes a local volume or directory as a share point. (Deprecated. There is no replacement function.)

PBShareSync (page 769) Deprecated in Mac OS X v10.4
: Establishes a local volume or directory as a share point. (Deprecated. There is no replacement function.)

PBUnshareAsync (page 773) Deprecated in Mac OS X v10.4
: Makes a share point unavailable on the network. (Deprecated. There is no replacement function.)

PBUnshareSync (page 773) Deprecated in Mac OS X v10.4
: Makes a share point unavailable on the network. (Deprecated. There is no replacement function.)

Functions

Allocate

Allocates additional space on a volume to an open file. (Deprecated in Mac OS X v10.4. Use FSAllocateFork (page 470) instead.)

```
OSErr Allocate (
   FSIORefNum refNum,
   SInt32 *count
);
```

Parameters

refNum
: The file reference number of the open file.

count
: On input, a pointer to the number of additional bytes to allocate to the file. On return, a pointer to the number of bytes actually allocated, rounded up to the nearest multiple of the allocation block size.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The `Allocate` function adds the specified number of bytes to the file and sets the physical end-of-file to 1 byte beyond the last block allocated. If there isn't enough empty space on the volume to satisfy the allocation request, `Allocate` allocates the rest of the space on the volume and returns `dskFulErr` as its function result.

The `Allocate` function always attempts to allocate contiguous blocks. If the total number of requested bytes is unavailable, `Allocate` allocates whatever space, contiguous or not, is available. To force the allocation of the entire requested space as a contiguous piece, call `AllocContig` (page 461) instead.

The File Manager automatically allocates file blocks if you move the logical end-of-file past the physical end-of-file, and it automatically deallocates unneeded blocks from a file if you move the logical end-of-file to a position more than one allocation block before the current physical end-of-file. Consequently, you do not in general need to be concerned with allocating or deallocating file blocks. However, you can improve file block contiguity if you use the `Allocate` or `AllocContig` function to preallocate file blocks. This is most useful if you know in advance how big a file is likely to become.

When the File Manager allocates (or deallocates) file blocks automatically, it always adds (or removes) blocks in clumps. The `Allocate` function allows you to add blocks in allocation blocks, which may be smaller than clumps.

The space allocated with this function is not permanently assigned to the file until the file's logical end-of-file is changed to include the allocated space. When a file (or volume) is closed, the space beyond the file's logical EOF is made available for other purposes, even if previously allocated to the file with a call to this function. You can change the end-of-file by setting it with the `SetEOF` (page 786) function, or by writing data to the file with the `FSWrite` (page 546) function.

This function is not supported by all file systems; for example, volumes mounted by the AppleShare file system do not support this function. To allocate space for a file on any volume, use the `SetEOF` (page 786) function, or one of the related parameter block calls, `PBSetEOFSync` (page 758) and `PBSetEOFAsync` (page 757).

To allocate space for a file beyond 2 GB, use the `FSAllocateFork` (page 470) function, or one of the corresponding parameter block functions, `PBAllocateForkSync` (page 567) and `PBAllocateForkAsync` (page 565).

Special Considerations

In Mac OS 7.5.5 through Mac OS 8.5, if there is not enough space left on the volume to allocate the requested number of bytes, the `Allocate` function does not return the number of bytes actually allocated. Your application should not rely on the value returned in the `count` parameter.

To determine the remaining space on a volume before calling `Allocate`, use the functions `PBXGetVolInfoSync` or `PBXGetVolInfoAsync`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

AllocContig

Allocates additional contiguous space on a volume to an open file. (Deprecated in Mac OS X v10.4. Use FSAllocateFork (page 470) instead.)

```
OSErr AllocContig (
   FSVolumeRefNum refNum,
   SInt32 *count
);
```

Parameters

refNum

The file reference number of an open file.

count

On input, a pointer to the number of additional bytes to allocate to the file; on return, a pointer to the number of bytes allocated, rounded up to the nearest multiple of the allocation block size.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The AllocContig function is identical to the Allocate (page 459) function except that if there isn't enough contiguous empty space on the volume to satisfy the allocation request, AllocContig does nothing and returns dskFulErr as its function result. If you want to allocate whatever space is available, even when the entire request cannot be filled by the allocation of a contiguous piece, call Allocate (page 459) instead.

The File Manager automatically allocates file blocks if you move the logical end-of-file past the physical end-of-file, and it automatically deallocates unneeded blocks from a file if you move the logical end-of-file to a position more than one allocation block before the current physical end-of-file. Consequently, you do not in general need to be concerned with allocating or deallocating file blocks. However, you can improve file block contiguity if you use the AllocContig function to preallocate file blocks. This is most useful if you know in advance how big a file is likely to become.

When the File Manager allocates (or deallocates) file blocks automatically, it always adds (or removes) blocks in clumps. The AllocContig function allows you to add blocks in allocation blocks, which may be smaller than clumps.

The space allocated with this function is not permanently assigned to the file until the file's logical end-of-file is changed to include the allocated space. When a file (or volume) is closed, the space beyond the file's logical EOF is made available for other purposes, even if previously allocated to the file with a call to this function. You can change the end-of-file by setting it with the SetEOF (page 786) function, or by writing data to the file with the FSWrite (page 546) function.

This function is not supported by all file systems; for example, volumes mounted by the AppleShare file system do not support this function. To allocate space for a file on any volume, use the SetEOF (page 786) function, or one of the related parameter block calls, PBSetEOFSync (page 758) and PBSetEOFAsync (page 757).

To allocate space for a file beyond 2 GB, use the FSAllocateFork (page 470) function, or one of the corresponding parameter block functions, PBAllocateForkSync (page 567) and PBAllocateForkAsync (page 565).

Special Considerations

In Mac OS 7.5.5 through Mac OS 8.5, when there is not enough space to allocate the requested number of bytes, `AllocContig` does not return 0 in the `count` parameter, so your application cannot rely upon this 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

`Files.h`

CatMove

Moves files or directories from one directory to another on the same volume. (Deprecated in Mac OS X v10.4. Use `FSMoveObject` (page 510) instead.)

```
OSErr CatMove (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param oldName,
   SInt32 newDirID,
   ConstStr255Param newName
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

The parent directory ID of the file or directory to move.

oldName

The existing name of the file or directory to move.

newDirID

If the `newName` parameter is empty, the directory ID of the destination directory; otherwise, the parent directory ID of the destination directory.

newName

The name of the destination directory. If a valid directory name is provided in this parameter, the destination directory's parent directory is specified in the `newDirID` parameter. However, you can specify an empty name for `newName`, in which case `newDirID` should be set to the directory ID of the destination directory.

It is usually simplest to specify the destination directory by passing its directory ID in the `newDirID` parameter and by setting `newName` to an empty name. To specify an empty name, set `newName` to `':'`.

Return Value

A result code. See "File Manager Result Codes" (page 943). This function returns `permErr` if called on a locked file.

Discussion
`CatMove` is strictly a file catalog operation; it does not actually change the location of the file or directory on the disk.

The `CatMove` function cannot move a file or directory to another volume (that is, the `vRefNum` parameter is used in specifying both the source and the destination). Also, you cannot use it to rename files or directories; to rename a file or directory, use `HRename` (page 555).

If you need to move files or directories with named forks other than the data and resource forks, with long Unicode names, or files larger than 2GB, you should use the `FSMoveObject` (page 510) function, or one of the corresponding parameter block calls, `PBMoveObjectSync` (page 738) and `PBMoveObjectAsync` (page 737).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

DirCreate

Creates a new directory. (Deprecated in Mac OS X v10.4. Use `FSCreateDirectoryUnicode` (page 479) instead.)

```
OSErr DirCreate (
   FSVolumeRefNum vRefNum,
   SInt32 parentDirID,
   ConstStr255Param directoryName,
   SInt32 *createdDirID
);
```

Parameters

vRefNum
A volume reference number, drive number, or 0 for the default volume.

parentDirID
The directory ID of the parent directory. If the parent directory ID is 0 and the volume specified in the `vRefNum` parameter is the default volume, the new directory is placed in the default directory of the volume. If the parent directory ID is 0 and the volume specified in the `vRefNum` parameter is a volume other than the default volume, the new directory is placed in the root directory of the volume. To create a directory at the root of a volume, regardless of whether that volume is the current default volume, pass the constant `fsRtDirID`(2) in this parameter.

directoryName
The name of the new directory.

createdDirID
On return, a pointer to the directory ID of the new directory. Note that a directory ID, unlike a volume reference number, is a long integer.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The date and time of the new directory's creation and last modification are set to the current date and time.

To create a directory with a Unicode name, use the function `FSCreateDirectoryUnicode` (page 479) , or one of the corresponding parameter block calls, `PBCreateDirectoryUnicodeSync` (page 589) and `PBCreateDirectoryUnicodeAsync` (page 587).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

DisposeFNSubscriptionUPP

Deletes a universal procedure pointer (UPP) to your directory change callback function.

```
void DisposeFNSubscriptionUPP (
   FNSubscriptionUPP userUPP
);
```

Parameters
userUPP
The UPP to delete.

Discussion
You should use this function to delete the UPP after the File Manager is finished calling your directory change callback function.

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

Declared In
`Files.h`

DisposeFSVolumeEjectUPP

Deletes a universal procedure pointer (UPP) to your volume ejection callback function.

```
void DisposeFSVolumeEjectUPP (
   FSVolumeEjectUPP userUPP
);
```

Parameters
userUPP
The UPP to delete.

Discussion
You should use this function to delete the UPP after the File Manager is finished calling your volume ejection callback function.

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

Declared In
`Files.h`

DisposeFSVolumeMountUPP

Deletes a universal procedure pointer (UPP) to your volume mount callback function.

```
void DisposeFSVolumeMountUPP (
   FSVolumeMountUPP userUPP
);
```

Parameters

userUPP

The UPP to delete.

Discussion
You should use this function to delete the UPP after the File Manager is finished calling your volume mount callback function.

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

Declared In
`Files.h`

DisposeFSVolumeUnmountUPP

Deletes a universal procedure pointer (UPP) to your volume unmount callback function.

```
void DisposeFSVolumeUnmountUPP (
   FSVolumeUnmountUPP userUPP
);
```

Parameters

userUPP

The UPP to delete.

Discussion
You should use this function to delete the UPP after the File Manager is finished calling your volume unmount callback function.

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

Declared In
`Files.h`

DisposeIOCompletionUPP

Deletes a universal procedure pointer (UPP) to your I/O completion callback function.

```
void DisposeIOCompletionUPP (
   IOCompletionUPP userUPP
);
```

Parameters

userUPP

The UPP to delete.

Discussion
You should use this function to delete the UPP after the File Manager is finished calling your I/O completion callback function.

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

Declared In
`Files.h`

FlushVol

Writes the contents of the volume buffer and updates information about the volume. (Deprecated in Mac OS X v10.5. Use `FSFlushVolume` (page 491) instead.)

```
OSErr FlushVol (
   ConstStr63Param volName,
   FSVolumeRefNum vRefNum
);
```

Parameters

volName

The name of the mounted volume to flush.

vRefNum

The volume reference number, drive number, or 0 for the default volume.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
For the specified volume, the `FlushVol` function writes the contents of the associated volume buffer and descriptive information about the volume. Information which has changed since the last time `FlushVol` was called is written to the volume.

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
`Files.h`

FNGetDirectoryForSubscription

Fetches the directory for which this subscription was originally entered.

```
OSStatus FNGetDirectoryForSubscription (
   FNSubscriptionRef subscription,
   FSRef *ref
);
```

Parameters

subscription

The subscription previously returned from the functions `FNSubscribe` or `FNSubscribeByPath`.

ref

On return, a file system reference to the directory for which this subscription was created.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

There is no path variant because paths are fragile, and the path may have changed. If the caller does not care about this subtlety, she can call `FSRefMakePath` to get a path from the returned reference.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`Files.h`

FNNotify

Broadcasts notification of changes to the specified directory.

```
OSStatus FNNotify (
   const FSRef *ref,
   FNMessage message,
   OptionBits flags
);
```

Parameters

ref

A file system reference describing the directory for which to broadcast the notification.

message

An indication of what happened to the target directory.

flags

Options regarding the delivery of the notification. Specify `kNilOptions` for the default behavior.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

FNNotifyAll

Broadcasts notification of changes to the filesystem.

```
OSStatus FNNotifyAll (
   FNMessage message,
   OptionBits flags
);
```

Parameters

message

An indication of what happened.

flags

Options regarding the delivery of the notification. Specify `kNilOptions` for the default behavior.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

This function should only be used by installers or programs which make lots of changes and only send one broadcast.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

FNNotifyByPath

Broadcasts notification of changes to the specified directory.

```
OSStatus FNNotifyByPath (
   const UInt8 *path,
   FNMessage message,
   OptionBits flags
);
```

Parameters

path

The path to the directory for which to broadcast the notification.

message

An indication of what happened to the target directory.

flags

Options regarding the delivery of the notification. Specify `kNilOptions` for the default behavior.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

FNSubscribe

Subscribes to change notifications for the specified directory.

```
OSStatus FNSubscribe (
   const FSRef *directoryRef,
   FNSubscriptionUPP callback,
   void *refcon,
   OptionBits flags,
   FNSubscriptionRef *subscription
);
```

Parameters

directoryRef

A file system reference describing the directory for which the caller wants notifications.

callback

A pointer to the function to call when a notification arrives.

refcon

A pointer to user state carried with the subscription.

flags

Specify `kNilOptions`, or one of the options described in "Notification Subscription Options" (page 925).

subscription

A subscription token for subsequent query or unsubscription.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Availability

Available in Mac OS X v10.1 and later.

Declared In

`Files.h`

FNSubscribeByPath

Subscribes to change notifications for the specified directory.

```
OSStatus FNSubscribeByPath (
   const UInt8 *directoryPath,
   FNSubscriptionUPP callback,
   void *refcon,
   OptionBits flags,
   FNSubscriptionRef *subscription
);
```

Parameters

directoryPath

A path to the directory for which the caller wants notifications.

callback

The function to call when a notification arrives.

refcon

A pointer to the user state carried with the subscription.

flags

Specify `kNilOptions`, or one of the options described in "Notification Subscription Options" (page 925).

subscription

A subscription token for subsequent query or unsubscription.

Return Value
A result code. See "File Manager Result Codes" (page 943).

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

Declared In
`Files.h`

FNUnsubscribe

Releases a subscription which is no longer needed.

```
OSStatus FNUnsubscribe (
   FNSubscriptionRef subscription
);
```

Parameters

subscription

A subscription previously returned from the `FNSubscribe` or`FNSubscribeByPath` functions.

Return Value
A result code. See "File Manager Result Codes" (page 943).

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

Declared In
`Files.h`

FSAllocateFork

Allocates space on a volume to an open fork.

```
OSErr FSAllocateFork (
   FSIORefNum forkRefNum,
   FSAllocationFlags flags,
   UInt16 positionMode,
   SInt64 positionOffset,
   UInt64 requestCount,
   UInt64 *actualCount
);
```

Parameters

forkRefNum

The reference number of the open fork. You can obtain a fork reference number with the `FSOpenFork` (page 514) function, or with one of the corresponding parameter block calls, `PBOpenForkSync` (page 740) and `PBOpenForkAsync` (page 739).

flags

A constant indicating how the new space should be allocated. See "Allocation Flags" (page 887) for a description of the constants which you can use in this parameter.

positionMode

A constant specifying the base location for the start of the allocation. See "Position Mode Constants" (page 928) for more information on the constants which you can use to specify the base location.

positionOffset

The offset from the base location of the start of the allocation.

requestCount

The number of bytes to allocate.

actualCount

On return, a pointer to the number of bytes actually allocated to the file. The value returned in here may be smaller than the number specified in the `requestCount` parameter if some of the space was already allocated. The value pointed to by the `actualCount` parameter does not reflect any additional bytes that may have been allocated because space is allocated in terms of fixed units such as allocation blocks, or the use of a clump size to reduce fragmentation.

The `actualCount` output is optional if you don't want the number of allocated bytes returned, set `actualCount` to `NULL`.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The `FSAllocateFork` function attempts to allocate `requestCount` bytes of physical storage starting at the offset specified by the `positionMode` and `positionOffset` parameters. For volume formats that support preallocated space, you can later write to this range of bytes (including extending the size of the fork) without requiring an implicit allocation.

Any extra space allocated but not used will be deallocated when the fork is closed, using `FSCloseFork` (page 475) , `PBCloseForkSync` (page 583) , or `PBCloseForkAsync` (page 582) ; or when the fork is flushed, using `FSFlushFork` (page 490) , `PBFlushForkSync` (page 639) , or `PBFlushForkAsync` (page 638).

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

Declared In
`Files.h`

FSCancelVolumeOperation

Cancels an outstanding asynchronous volume mounting operation.

```
OSStatus FSCancelVolumeOperation (
   FSVolumeOperation volumeOp
);
```

Parameters

volumeOp

The asynchronous volume operation to cancel.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Special Considerations

This function currently is only supported for server mounts.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Files.h`

FSCatalogSearch

Searches for objects traversed by a catalog iterator that match a given set of criteria.

```
OSErr FSCatalogSearch (
   FSIterator iterator,
   const FSSearchParams *searchCriteria,
   ItemCount maximumObjects,
   ItemCount *actualObjects,
   Boolean *containerChanged,
   FSCatalogInfoBitmap whichInfo,
   FSCatalogInfo *catalogInfos,
   FSRef *refs,
   FSSpecPtr specs,
   HFSUniStr255 *names
);
```

Parameters

iterator

The iterator to use. Objects traversed by this iterator are matched against the criteria specified by the `searchCriteria` parameter. You can obtain a catalog iterator with the function `FSOpenIterator` (page 515), or with one of the related parameter block calls, `PBOpenIteratorSync` (page 742) and `PBOpenIteratorAsync` (page 741). Currently, this iterator must be created with the `kFSIterateSubtree` option and the container must be the root directory of a volume. See `FSIterator` (page 835) for more information on the `FSIterator` data type.

searchCriteria

A pointer to a structure containing the search criteria.

You can match against the object's name in Unicode and by the fields in an `FSCatalogInfo` (page 826) structure. You may use the same search bits as passed in the `ioSearchBits` field to the `PBCatSearchSync` (page 580) and `PBCatSearchAsync` (page 577) functions; they control the corresponding `FSCatalogInfo` fields. See "Catalog Search Masks" (page 900) for a description of the search bits.

There are a few new search criteria supported by `FSCatalogSearch` but not by `PBCatSearchSync` and `PBCatSearchAsync`. These new search criteria are indicated by the constants described in "Catalog Search Constants" (page 899).

If the `searchTime` field of this structure is non-zero, it is interpreted as a Time Manager duration; the search may terminate after this duration even if `maximumObjects` objects have not been returned and the entire catalog has not been scanned. If `searchTime` is zero, there is no time limit for the search.

If you are searching by any criteria other than name, you must set the `searchInfo1` and `searchInfo2` fields of the structure in this parameter to point to `FSCatalogInfo` structures containing the values to match against.

See `FSSearchParams` (page 839) for a description of the `FSSearchParams` data type.

maximumObjects

The maximum number of items to return for this call.

actualObjects

On return, a pointer to the actual number of items found for this call.

containerChanged

On return, a pointer to a Boolean value indicating whether the container's contents have changed. If `true`, the container's contents changed since the previous `FSCatalogSearch` call. Objects may still be returned even though the container changed. Note that if the container has changed, then the total set of items returned may be incorrect; some items may be returned multiple times, and some items may not be returned at all.

This parameter is optional if you don't want this information, pass a `NULL` pointer.

whichInfo

A bitmap specifying the catalog information fields to return for each item. If you don't wish any catalog information returned, pass the constant `kFSCatInfoNone` in this parameter. See "Catalog Information Bitmap Constants" (page 891) for a description of the bits in this parameter.

catalogInfos

A pointer to an array of catalog information structures; one for each found item. On input, the `catalogInfos` parameter should point to an array of `maximumObjects` catalog information structures.

This parameter is optional; if you do not wish any catalog information returned, pass `NULL` here.

See `FSCatalogInfo` (page 826) for a description of the `FSCatalogInfo` data type.

refs

A pointer to an array of `FSRef` structures; one for each returned item. If you want an `FSRef` for each item found, set this parameter to point to an array of `maximumObjectsFSRef` structures. Otherwise, set it to `NULL`. See `FSRef` (page 837) for a description of the `FSRef` data type.

specs
names

A pointer to an array of filenames; one for each returned item. If you want the Unicode filename for each item found, set this parameter to point to an array of `maximumObjectsHFSUniStr255` structures. Otherwise, set it to `NULL`. See `HFSUniStr255` (page 855) for a description of the `HFSUniStr255` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943). When the entire volume has been searched, `errFSNoMoreItems` is returned.

Discussion
A single search may span more than one call to `FSCatalogSearch`. The call may complete with no error before scanning the entire volume. This typically happens because the time limit (`searchTime`) has been reached or `maximumObjects` items have been returned. If the search is not completed, you can continue the search by making another call to `FSCatalogSearch` and passing the updated iterator returned by the previous call in the `iterator` parameter.

Before calling this function, you should determine that it is present, by calling the `Gestalt` function.

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

Declared In
`Files.h`

FSClose

Closes an open file. (Deprecated in Mac OS X v10.4. Use `FSCloseFork` (page 475) instead.)

```
OSErr FSClose (
    FSIORefNum refNum
);
```

Parameters

refNum

The file reference number of the open file.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The `FSClose` function removes the access path for the specified file and writes the contents of the volume buffer to the volume.

The `FSClose` function calls the `PBFlushFileSync` function internally to write the file's bytes onto the volume. To ensure that the file's catalog entry is updated, you should call `FlushVol` (page 466) after you call `FSClose`.

Special Considerations

Make sure that you do not call `FSClose` with a file reference number of a file that has already been closed. Attempting to close the same file twice may result in loss of data on a volume.

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

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

FSCloseFork

Closes an open fork.

```
OSErr FSCloseFork (
    FSIORefNum forkRefNum
);
```

Parameters

forkRefNum

The reference number of the fork to close. After the call to this function, the reference number in this parameter is invalid.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The `FSCloseFork` function causes all data written to the fork to be written to disk, in the same manner as the `FSFlushFork` (page 490) function, before it closes the fork.

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

Declared In
`Files.h`

FSCloseIterator

Closes a catalog iterator.

```
OSErr FSCloseIterator (
   FSIterator iterator
);
```

Parameters

iterator

The catalog iterator to be closed. `FSCloseIterator` releases memory and other system resources used by the iterator, making the iterator invalid. See `FSIterator` (page 835) for a description of the `FSIterator` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
This function releases memory and other system resources used by the iterator. The iterator becomes invalid.

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

Related Sample Code
QTCarbonShell

Declared In
`Files.h`

FSCompareFSRefs

Determines whether two `FSRef` structures refer to the same file or directory.

```
OSErr FSCompareFSRefs (
   const FSRef *ref1,
   const FSRef *ref2
);
```

Parameters

ref1

A pointer to the first `FSRef` to compare. For a description of the `FSRef` data type, see `FSRef` (page 837).

ref2

A pointer to the second `FSRef` to compare.

Return Value
A result code. See "File Manager Result Codes" (page 943). If the two `FSRef` structures refer to the same file or directory, then `noErr` is returned. If they refer to objects on different volumes, then `diffVolErr` is returned. If they refer to different files or directories on the same volume, then `errFSRefsDifferent` is returned. This function may return other errors, including `nsvErr`, `fnfErr`, `dirNFErr`, and `volOffLinErr`.

Discussion
You must use `FSCompareFSRefs`, or one of the corresponding parameter block functions, `PBCompareFSRefsSync` (page 586) and `PBCompareFSRefsAsync` (page 586) , to compare `FSRef` structures. It is not possible to compare the `FSRef` structures directly since some bytes may be uninitialized, case-insensitive text, or contain hint information.

Some volume formats may be able to tell that two `FSRef` structures would refer to two different files or directories, without having to actually find those objects. In this case, the volume format may return `errFSRefsDifferent` even if one or both objects no longer exist. Similarly, if the `FSRef` structures are for objects on different volumes, the File Manager will return `diffVolErr` even if one or both volumes are no longer mounted.

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

Declared In
`Files.h`

FSCopyDiskIDForVolume

Returns a copy of the disk ID for a volume.

```
OSStatus FSCopyDiskIDForVolume (
    FSVolumeRefNum vRefNum,
    CFStringRef *diskID
);
```

Parameters

vRefNum

The volume reference number of the target volume.

diskID

A pointer to a Core Foundation string. On return, the string contains the disk ID associated with the target volume. The caller is responsible for releasing the string.

Return Value
A result code. See "File Manager Result Codes" (page 943).

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

Declared In
`Files.h`

FSCopyObjectAsync

Starts an asynchronous file operation to copy a source object to a destination directory.

```
OSStatus FSCopyObjectAsync (
   FSFileOperationRef fileOp,
   const FSRef *source,
   const FSRef *destDir,
   CFStringRef destName,
   OptionBits flags,
   FSFileOperationStatusProcPtr callback,
   CFTimeInterval statusChangeInterval,
   FSFileOperationClientContext *clientContext
);
```

Parameters

fileOp

The file operation object you created for this copy operation.

source

A pointer to the source object to copy. The object can be a file or a directory.

destDir

A pointer to the destination directory.

destName

The name for the new object in the destination directory. Pass `NULL` to use the name of the source object.

flags

One or more file operation option flags. See "File Operation Options" (page 917).

callback

A callback function to receive status updates as the file operation proceeds. For more information, see "File Operation Callbacks" (page 788). This parameter is optional; pass `NULL` if you don't need to supply a status callback.

statusChangeInterval
The minimum time in seconds between callbacks within a single stage of an operation.

clientContext
User-defined data to associate with this operation. For more information, see `FSFileOperationClientContext` (page 829). This parameter is optional; pass `NULL` if you don't need to supply a client context.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
If you specify a status callback function, status callbacks will occur in one of the run loop and mode combinations with which you scheduled the file operation.

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

Declared In
`Files.h`

FSCopyObjectSync

Copies a source object to a destination directory.

```
OSStatus FSCopyObjectSync (
   const FSRef *source,
   const FSRef *destDir,
   CFStringRef destName,
   FSRef *target,
   OptionBits options
);
```

Parameters

source
A pointer to the source object to copy. The object can be a file or a directory.

destDir
A pointer to the destination directory.

destName
The name for the new object in the destination directory. Pass `NULL` to use the name of the source object.

target
A pointer to an `FSRef` variable that, on output, refers to the new object in the destination directory. This parameter is optional; pass `NULL` if you don't need to refer to the new object.

options
One or more file operation option flags. See "File Operation Options" (page 917).

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
This function could take a significant amount of time to execute. To avoid blocking your user interface, you should either call this function in a thread other than the main thread or use `FSCopyObjectAsync` (page 477) instead.

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

Declared In
`Files.h`

FSCopyURLForVolume

Returns a copy of the URL for a volume.

```
OSStatus FSCopyURLForVolume (
    FSVolumeRefNum vRefNum,
    CFURLRef *url
);
```

Parameters

vRefNum

The volume reference number of the target volume.

url

A pointer to a `CFURLRef` variable allocated by the caller. On return, a Core Foundation URL that specifies the location of the target volume. The caller is responsible for releasing the URL.

Return Value
A result code. See "File Manager Result Codes" (page 943).

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

Declared In
`Files.h`

FSCreateDirectoryUnicode

Creates a new directory (folder) with a Unicode name.

```
OSErr FSCreateDirectoryUnicode (
   const FSRef *parentRef,
   UniCharCount nameLength,
   const UniChar *name,
   FSCatalogInfoBitmap whichInfo,
   const FSCatalogInfo *catalogInfo,
   FSRef *newRef,
   FSSpecPtr newSpec,
   UInt32 *newDirID
);
```

Parameters

parentRef

A pointer to an `FSRef` specifying the parent directory where the new directory is to be created. See `FSRef` (page 837) for a description of the `FSRef` data type.

nameLength

The length of the new directory's Unicode name.

name

A pointer to the Unicode name of the new directory.

whichInfo

A bitmap specifying which catalog information fields to set for the new directory. Specify the values for these fields in the `catalogInfo` parameter.

If you do not wish to set catalog information for the new directory, specify the constant `kFSCatInfoNone`. See "Catalog Information Bitmap Constants" (page 891) for a description of the bits defined for this parameter.

catalogInfo

A pointer to the `FSCatalogInfo` structure which specifies the values for the catalog information fields for the new directory. Specify which fields to set in the `whichInfo` parameter.

This parameter is optional; specify `NULL` if you do not wish to set catalog information for the new directory.

See `FSCatalogInfo` (page 826) for a description of the `FSCatalogInfo` data type.

newRef

On return, a pointer to the `FSRef` for the new directory. This parameter is optional; specify `NULL` if you do not want the `FSRef` returned.

newSpec

On return, a pointer to the `FSSpec` for the new directory. This parameter is optional; specify `NULL` if you do not want the `FSSpec` returned. See `FSSpec` (page 840) for a description of the `FSSpec` data type.

newDirID

On return, a pointer to the directory ID of the directory. This parameter is optional; specify `NULL` if you do not want the directory ID returned.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
You may optionally set catalog information for the new directory using the `whichInfo` and `catalogInfo` parameters; this is equivalent to calling `FSSetCatalogInfo` (page 540) , or one of the corresponding parameter block functions, `PBSetCatalogInfoSync` (page 753) and `PBSetCatalogInfoAsync` (page 751) , after creating the directory.

If possible, you should set the `textEncodingHint` field of the catalog information structure specified in the `catalogInfo` parameter. This will be used by the volume format when converting the Unicode filename to other encodings.

Special Considerations

If the `FSCreateDirectoryUnicode` function is present, but is not implemented by a particular volume, the File Manager will emulate this function by making the appropriate call to `PBDirCreateSync` (page 601). However, if the function is not directly supported by the volume, you will not be able to use the long Unicode directory names, or other features added with HFS Plus.

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

Related Sample Code
BSDLLCTest

Declared In
`Files.h`

FSCreateFileUnicode

Creates a new file with a Unicode name.

```
OSErr FSCreateFileUnicode (
   const FSRef *parentRef,
   UniCharCount nameLength,
   const UniChar *name,
   FSCatalogInfoBitmap whichInfo,
   const FSCatalogInfo *catalogInfo,
   FSRef *newRef,
   FSSpecPtr newSpec
);
```

Parameters

parentRef
A pointer to an `FSRef` for the directory where the file is to be created. See `FSRef` (page 837) for a description of the `FSRef` data type.

nameLength
The length of the file's name.

name
A pointer to the Unicode name for the new file.

whichInfo
A bitmap specifying which catalog information fields to set for the new file. You specify the values for these fields in the `catalogInfo` parameter. If you do not wish to set catalog information for the new file, pass the constant `kFSCatInfoNone`. See "Catalog Information Bitmap Constants" (page 891) for a description of the bits defined for this parameter.

catalogInfo
A pointer to the `FSCatalogInfo` structure which specifies the values of the new file's catalog information. Specify which fields to set in the `whichInfo` parameter.

This parameter is optional; specify `NULL` if you do not wish to set catalog information for the new file.

newRef
On return, a pointer to the `FSRef` for the new file. If you do not want the `FSRef` returned, specify `NULL`.

newSpec
On return, a pointer to the `FSSpec` for the new file. If you do not want the `FSSpec` returned, specify `NULL`. See `FSSpec` (page 840) for a description of the `FSSpec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
You may optionally set catalog information for the new file using the `whichInfo` and `catalogInfo` parameters; this is equivalent to calling `FSSetCatalogInfo` (page 540) , or one of the corresponding parameter block functions, `PBSetCatalogInfoSync` (page 753) and `PBSetCatalogInfoAsync` (page 751) , after creating the file.

If possible, you should set the `textEncodingHint` field of the catalog information structure specified in the `catalogInfo` parameter. This will be used by the volume format when converting the Unicode filename to other encodings.

Special Considerations

If the `FSCreateFileUnicode` function is present, but is not implemented by a particular volume, the File Manager will emulate this function by making the appropriate call to `PBHCreateSync` (page 677). However, if the function is not directly supported by the volume, you will not be able to use the long Unicode filenames, or other features added with HFS Plus.

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

Related Sample Code
BSDLLCTest
CarbonSketch
QTCarbonShell

Declared In
`Files.h`

FSCreateFork

Creates a named fork for a file or directory.

```
OSErr FSCreateFork (
   const FSRef *ref,
   UniCharCount forkNameLength,
   const UniChar *forkName
);
```

Parameters

ref
A pointer to an `FSRef` specifying the file or directory. See `FSRef` (page 837) for a description of the `FSRef` data type.

forkNameLength
The length of the name of the new fork.

forkName
A pointer to the Unicode name of the fork.

Return Value
A result code. See "File Manager Result Codes" (page 943). If the named fork already exists, the function returns `errFSForkExists`. If the fork name is syntactically invalid or otherwise unsupported for the given volume, `FSCreateFork` returns `errFSBadForkName` or `errFSNameTooLong`.

Discussion
A newly created fork has zero length (that is, its logical end-of-file is zero). The data and resource forks of a file are automatically created and deleted as needed. This is done for compatibility with older APIs, and because data and resource forks are often handled specially. If a given fork always exists for a given volume format (such as data and resource forks for HFS and HFS Plus, or data forks for most other volume formats), an attempt to create that fork when a zero-length fork already exists should return `noErr`; if a non-empty fork already exists then `errFSForkExists` should be returned.

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

Declared In
`Files.h`

FSCreateVolumeOperation

Returns an `FSVolumeOperation` which can be used for an asynchronous volume operation.

```
OSStatus FSCreateVolumeOperation (
   FSVolumeOperation *volumeOp
);
```

Parameters

`volumeOp`
The new `FSVolumeOperation`.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
When the operation is completed the `FSVolumeOperation` should be disposed of to free the memory associated with the operation using `FSDisposeVolumeOperation`.

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

Declared In
`Files.h`

FSDeleteFork

Deletes a named fork from a file or directory.

```
OSErr FSDeleteFork (
   const FSRef *ref,
   UniCharCount forkNameLength,
   const UniChar *forkName
);
```

Parameters

`ref`
A pointer to an `FSRef` for the file or directory from which to delete the fork. See `FSRef` (page 837) for a description of the `FSRef` data type.

`forkNameLength`
The length of the Unicode name of the fork name.

`forkName`
A pointer to the Unicode name of the fork to delete.

Return Value
A result code. See "File Manager Result Codes" (page 943). If the named fork does not exist, the function returns `errFSForkNotFound`.

Discussion
Any storage allocated to the fork is released. If a given fork always exists for a given volume format (such as data and resource forks for HFS and HFS Plus, or data forks for most other volume formats), this is equivalent to setting the logical size of the fork to zero.

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

Declared In
`Files.h`

FSDeleteObject

Deletes a file or an empty directory.

```
OSErr FSDeleteObject (
   const FSRef *ref
);
```

Parameters

ref

A pointer to an `FSRef` specifying the file or directory to be deleted. If the object to be deleted is a directory, it must be empty (it must contain no files or folders). See `FSRef` (page 837) for a description of the `FSRef` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943). If you attempt to delete a folder for which there is an open catalog iterator, this function succeeds and returns `noErr`. Iteration, however, will continue to work until the iterator is closed.

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

Related Sample Code
BSDLLCTest
CarbonSketch
QTCarbonShell

Declared In
`Files.h`

FSDisposeVolumeOperation

Releases the memory associated with a volume operation.

```
OSStatus FSDisposeVolumeOperation (
   FSVolumeOperation volumeOp
);
```

Parameters

volumeOp

The `FSVolumeOperation` to release.

Return Value
A result code. See "File Manager Result Codes" (page 943). This function will return `paramErr` if the `FSVolumeOperation` is in use.

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

Declared In
`Files.h`

FSEjectVolumeAsync

Asynchronously ejects a volume.

```
OSStatus FSEjectVolumeAsync (
   FSVolumeRefNum vRefNum,
   OptionBits flags,
   FSVolumeOperation volumeOp,
   void *clientData,
   FSVolumeEjectUPP callback,
   CFRunLoopRef runloop,
   CFStringRef runloopMode
);
```

Parameters

vRefNum
The volume reference number of the volume to eject.

flags
Options for future use.

volumeOp
An `FSVolumeOperation` returned by `FSCreateVolumeOperation`.

clientData
A pointer to client data associated with the operation. This parameter can be `NULL`.

callback
The function to call when eject is complete.

runloop
The runloop to run on.

runloopMode
The mode for the runloop.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
This function starts the process of ejecting the volume specified by the *vRefNum* parameter. If a callback function is provided, that function will be called when the eject operation is complete. Once this function returns `noErr` the status of the operation can be found using `FSGetAsyncEjectStatus`.

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

Declared In
Files.h

FSEjectVolumeSync

Ejects a volume.

```
OSStatus FSEjectVolumeSync (
   FSVolumeRefNum vRefNum,
   OptionBits flags,
   pid_t *dissenter
);
```

Parameters

vRefNum
The volume reference number of the volume to eject.

flags
Options for future use.

dissenter
On return, a pointer to the pid of the process which denied the unmount if the eject is denied.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
This function ejects the volume specified by the *vRefNum* parameter. If the volume cannot be ejected the pid of the process which denied the unmount will be returned in the *dissenter* parameter. This function returns after the eject is complete. Ejecting a volume will result in the unmounting of other volumes on the same device.

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

Declared In
Files.h

FSExchangeObjects

Swaps the contents of two files.

```
OSErr FSExchangeObjects (
   const FSRef *ref,
   const FSRef *destRef
);
```

Parameters

ref
A pointer to an FSRef for the first file. See FSRef (page 837) for a description of the FSRef data type.

destRef
A pointer to an FSRef for the second file.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The `FSExchangeObjects` function allows programs to implement a "safe save" operation by creating and writing a complete new file and swapping the contents. An alias, `FSSpec`, or `FSRef` that refers to the old file will now access the new data. The corresponding information in in-memory data structures are also exchanged.

Either or both files may have open access paths. After the exchange, the access path will refer to the opposite file's data (that is, to the same data it originally referred, which is now part of the other file).

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

Declared In
`Files.h`

FSFileOperationCancel

Cancels an asynchronous file operation.

```
OSStatus FSFileOperationCancel (
   FSFileOperationRef fileOp
);
```

Parameters

`fileOp`
The file operation to cancel.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
This function makes the specified file operation ineligible to run on any run loop. You may call this function at any time during the operation. Typically, you would use this function if the user cancels the operation. Note that to release your file operation object, you still need to call `CFRelease`.

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

Declared In
`Files.h`

FSFileOperationCopyStatus

Gets a copy of the current status information for an asynchronous file operation.

```
OSStatus FSFileOperationCopyStatus (
   FSFileOperationRef fileOp,
   FSRef *currentItem,
   FSFileOperationStage *stage,
   OSStatus *error,
   CFDictionaryRef *statusDictionary,
   void **info
);
```

Parameters

fileOp

The file operation to access.

currentItem

A pointer to an `FSRef` variable. On output, the variable contains the object currently being moved or copied. If the operation is complete, this parameter refers to the target (the new object corresponding to the source object in the destination directory).

stage

A pointer to a file operation stage variable. On output, the variable contains the current stage of the file operation.

error

A pointer to an error status variable. On output, the variable contains the current error status of the file operation.

statusDictionary

A pointer to a dictionary variable. On output, the variable contains a dictionary with more detailed status information. For information about the contents of the dictionary, see "`File Operation Status Dictionary Keys`" (page 919). You should release the dictionary when you are finished using it.

info

A pointer to a generic pointer. On output, the generic pointer refers to user-defined data associated with this file operation.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`Files.h`

FSFileOperationCreate

Creates an object that represents an asynchronous file operation.

```
FSFileOperationRef FSFileOperationCreate (
   CFAllocatorRef alloc
);
```

Parameters

alloc

The allocator to use. Pass `NULL` for the default allocator.

Return Value
A new FSFileOperation object, or `NULL` if the object could not be created. When you no longer need the object, you should release it by calling `CFRelease`.

Discussion
Before passing a file operation object to a function that starts an asynchronous copy or move operation, you should schedule the file operation using the function `FSFileOperationScheduleWithRunLoop` (page 489).

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

Declared In
`Files.h`

FSFileOperationGetTypeID

Returns the Core Foundation type identifier for the FSFileOperation opaque type.

```
CFTypeID FSFileOperationGetTypeID (
   void
);
```

Return Value
The type identifier for the FSFileOperation opaque type. For information about this type, see `FSFileOperationRef` (page 830).

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

Declared In
`Files.h`

FSFileOperationScheduleWithRunLoop

Schedules an asynchronous file operation with the specified run loop and mode.

```
OSStatus FSFileOperationScheduleWithRunLoop (
   FSFileOperationRef fileOp,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

fileOp
The file operation to schedule.

runLoop
The run loop in which to schedule the operation. For information about Core Foundation run loops, see *Run Loops*.

runLoopMode
The run loop mode in which to schedule the operation. In most cases, you may specify `kCFRunLoopCommonModes`.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
To run, a file operation must be scheduled with at least one run loop. A file operation can be scheduled with multiple run loop and mode combinations.

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

Declared In
`Files.h`

FSFileOperationUnscheduleFromRunLoop

Unschedules an asynchronous file operation from the specified run loop and mode.

```
OSStatus FSFileOperationUnscheduleFromRunLoop (
   FSFileOperationRef fileOp,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

fileOp
The file operation to unschedule.

runLoop
The run loop on which to unschedule the operation.

runLoopMode
The run loop mode in which to unschedule the operation.

Return Value
A result code. See "File Manager Result Codes" (page 943).

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

Declared In
`Files.h`

FSFlushFork

Causes all data written to an open fork to be written to disk.

```
OSErr FSFlushFork (
   FSIORefNum forkRefNum
);
```

Parameters

forkRefNum
The reference number of the fork to flush.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The `FSFlushFork` function causes the actual fork contents to be written to disk, as well as any other volume structures needed to access the fork. On HFS and HFS Plus, this includes the catalog, extents, and attribute B-trees; the volume bitmap; and the volume header and alternate volume header (the MDB and alternate MDB on HFS volumes), as needed.

On volumes that do not support `FSFlushFork` directly, the entire volume is flushed to be sure all volume structures associated with the fork are written to disk.

You do not, need to use `FSFlushFork` to flush a file fork before it is closed; the file is automatically flushed when it is closed and all cache blocks associated with it are removed from the cache.

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

Declared In
`Files.h`

FSFlushVolume

For the specified volume, writes all open and modified files in the current process to permanent storage.

```
OSStatus FSFlushVolume (
    FSVolumeRefNum vRefNum
);
```

Parameters

vRefNum
The volume reference number of the volume to flush.

Return Value
A result code. See "File Manager Result Codes" (page 943).

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

Declared In
`Files.h`

FSGetAsyncEjectStatus

Returns the current status of an asynchronous eject operation.

```
OSStatus FSGetAsyncEjectStatus (
   FSVolumeOperation volumeOp,
   FSEjectStatus *status,
   OSStatus *volumeOpStatus,
   FSVolumeRefNum *volumeRefNum,
   pid_t *dissenter,
   void **clientData
);
```

Parameters

volumeOp

The asynchronous volume operation to get status about.

status

On return, a pointer to the status of the operation.

volumeOpStatus

If the *status* parameter is `kAsyncEjectComplete` then this contains the result code (`OSStatus`) for the operation on return.

volumeRefNum

On return, the volume reference number of the volume being ejected.

dissenter

On return, a pointer to the pid of the process which denied the unmount if the eject is denied.

clientData

On return, a pointer to client data associated with the original `FSMountServerVolumeAsync` operation.

Return Value

A result code. See "File Manager Result Codes" (page 943). A return value of `noErr` signifies that the `status` parameter has been filled with valid information.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Files.h`

FSGetAsyncMountStatus

Returns the current status of an asynchronous mount operation.

```
OSStatus FSGetAsyncMountStatus (
   FSVolumeOperation volumeOp,
   FSMountStatus *status,
   OSStatus *volumeOpStatus,
   FSVolumeRefNum *mountedVolumeRefNum,
   void **clientData
);
```

Parameters

volumeOp

The asynchronous volume operation to get status about.

status

On return, a pointer to the status of the operation.

volumeOpStatus
If the status is kAsyncMountComplete then this parameter contains the result code for the operation on return.

mountedVolumeRefNum
If the status is kAsyncMountComplete and the *volumeOpStatus* parameter is noErr then this is the volume reference number for the newly mounted volume, on return.

clientData
On return, a pointer to client data associated with the original FSMountServerVolumeAsync operation.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
A return value of noErr signifies that the *status* parameter has been filled with valid information. If the status is kAsyncMountComplete then the rest of data returned is valid. If the status is anything else then the *volumeOpStatus* and *mountedVolumeRefNum* parameters are invalid, but the *clientData* parameter is valid.

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

Declared In
Files.h

FSGetAsyncUnmountStatus

Returns the current status of an asynchronous unmount operation.

```
OSStatus FSGetAsyncUnmountStatus (
   FSVolumeOperation volumeOp,
   FSUnmountStatus *status,
   OSStatus *volumeOpStatus,
   FSVolumeRefNum *volumeRefNum,
   pid_t *dissenter,
   void **clientData
);
```

Parameters

volumeOp
The asynchronous volume operation to get status about.

status
On return, a pointer to the status of the operation.

volumeOpStatus
If the status is kAsyncUnmountComplete then this parameter contains a pointer to the result code (OSStatus) for the operation on return.

volumeRefNum
On return, a pointer to the volume reference number of the volume being unmounted.

dissenter
On return, a pointer to the pid of the process which denied the unmount if the unmount is denied.

clientData
On return, a pointer to client data associated with the original `FSMountServerVolumeAsync` operation.

Return Value
A result code. See "File Manager Result Codes" (page 943). A return value of `noErr` signifies that the *status* parameter has been filled with valid information.

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

Declared In
`Files.h`

FSGetCatalogInfo

Returns catalog information about a file or directory. You can use this function to map an `FSRef` to an `FSSpec`.

```
OSErr FSGetCatalogInfo (
   const FSRef *ref,
   FSCatalogInfoBitmap whichInfo,
   FSCatalogInfo *catalogInfo,
   HFSUniStr255 *outName,
   FSSpecPtr fsSpec,
   FSRef *parentRef
);
```

Parameters

ref
A pointer to an `FSRef` specifying the file or directory for which to retrieve information. See `FSRef` (page 837) for a description of the `FSRef` data type.

whichInfo
A bitmap specifying the catalog information fields to return. If you don't want any catalog information, set `whichInfo` to the constant `kFSCatInfoNone`. See "Catalog Information Bitmap Constants" (page 891) for a description of the bits in this parameter.

catalogInfo
On return, a pointer to a catalog information structure containing the information about the file or directory. Only the information specified in the `whichInfo` parameter is returned. If you don't want any catalog information, pass `NULL` here. See `FSCatalogInfo` (page 826) for a description of the `FSCatalogInfo` data type.

outName
On return, a pointer to the Unicode name of the file or directory is returned here. This parameter is optional; if you do not wish the name returned, pass `NULL` here. See `HFSUniStr255` (page 855) for a description of the `HFSUniStr255` data type.

fsSpec
On return, a pointer to the `FSSpec` for the file or directory. This parameter is optional; if you do not wish the `FSSpec` returned, pass `NULL` here. See `FSSpec` (page 840) for a description of the `FSSpec` data type.

parentRef

On return, a pointer to the `FSRef` for the object's parent directory. This parameter is optional; if you do not wish the parent directory returned, pass `NULL` here.

If the object specified in the `ref` parameter is a volume's root directory, then the `FSRef` returned here will not be a valid `FSRef`, since the root directory has no parent object.

Return Value
A result code. See "File Manager Result Codes" (page 943).

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

Related Sample Code
BSDLLCTest
QTCarbonShell

Declared In
`Files.h`

FSGetCatalogInfoBulk

Returns information about one or more objects from a catalog iterator. This function can return information about multiple objects in a single call.

```
OSErr FSGetCatalogInfoBulk (
   FSIterator iterator,
   ItemCount maximumObjects,
   ItemCount *actualObjects,
   Boolean *containerChanged,
   FSCatalogInfoBitmap whichInfo,
   FSCatalogInfo *catalogInfos,
   FSRef *refs,
   FSSpecPtr specs,
   HFSUniStr255 *names
);
```

Parameters

iterator

The iterator to use. You can obtain a catalog iterator with the function `FSOpenIterator` (page 515), or with one of the related parameter block calls, `PBOpenIteratorSync` (page 742) and `PBOpenIteratorAsync` (page 741). Currently, the iterator must be created with the `kFSIterateFlat` option. See `FSIterator` (page 835) for a description of the `FSIterator` data type.

maximumObjects

The maximum number of items to return for this call.

actualObjects

On return, a pointer to the actual number of items found for this call.

containerChanged

On return, a pointer to a value indicating whether or not the container's contents have changed since the previous `FSGetCatalogInfoBulk` call. If `true`, the contents have changed. Objects may still be returned, even though the container has changed. If so, note that if the container has changed, then the total set of items returned may be incorrect: some items may be returned multiple times, and some items may not be returned at all.

This parameter is optional if you don't want this information returned, pass a `NULL` pointer.

In Mac OS X version 10.2 and later, this parameter is always set to `false`. To find out whether the container has changed since the last call to `FSGetCatalogInfoBulk`, check the modification date of the container.

whichInfo

A bitmap specifying the catalog information fields to return for each item. If you don't wish any catalog information returned, pass the constant `kFSCatInfoNone` in this parameter. For a description of the bits in this parameter, see "Catalog Information Bitmap Constants" (page 891).

catalogInfos

A pointer to an array of catalog information structures; one for each returned item. On input, the `catalogInfos` parameter should point to an array of `maximumObjects` catalog information structures.

This parameter is optional; if you do not wish any catalog information returned, pass `NULL` here.

refs

A pointer to an array of `FSRef` structures; one for each returned item. On input, this parameter should to point to an array of `maximumObjectsFSRef` structures.

This parameter is optional; if you do not wish any `FSRef` structures returned, pass `NULL` here.

specs

A pointer to an array of `FSSpec` structures; one for each returned item. On input, this parameter should to point to an array of `maximumObjectsFSSpec` structures.

This parameter is optional; if you do not wish any `FSSpec` structures returned, pass `NULL` here.

names

A pointer to an array of names; one for each returned item. If you want the Unicode name for each item found, set this parameter to point to an array of `maximumObjectsHFSUniStr255` structures. Otherwise, set it to `NULL`.

Return Value
A result code. See "File Manager Result Codes" (page 943). When all of the iterator's objects have been returned, the call will return `errFSNoMoreItems`.

Discussion
The `FSGetCatalogInfoBulk` call may complete and return `noErr` with fewer than `maximumObjects` items returned. This may be due to various reasons related to the internal implementation. In this case, you may continue to make `FSGetCatalogInfoBulk` calls using the same iterator.

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

Related Sample Code
QTCarbonShell

Declared In
`Files.h`

FSGetDataForkName

Returns a Unicode string constant for the name of the data fork.

```
OSErr FSGetDataForkName (
   HFSUniStr255 *dataForkName
);
```

Parameters

dataForkName

On input, a pointer to an `HFSUniStr255` structure. On return, this structure contains the Unicode name of the data fork. Currently, this is the empty string. See `HFSUniStr255` (page 855) for a description of the `HFSUniStr255` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

There is no parameter block-based form of this call since it is not dispatched to individual volume formats, and does not require any I/O.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

FSGetForkCBInfo

Returns information about a specified open fork, or about all open forks.

```
OSErr FSGetForkCBInfo (
   FSIORefNum desiredRefNum,
   FSVolumeRefNum volume,
   short *iterator,
   FSIORefNum *actualRefNum,
   FSForkInfo *forkInfo,
   FSRef *ref,
   HFSUniStr255 *outForkName
);
```

Parameters

desiredRefNum

If you want information on a specific fork, set this parameter to that fork's reference number, and pass `NULL` in the `iterator` parameter. If you pass a non-zero value in this parameter, the function attempts to get information on the fork specified by that reference number.

Pass zero in this parameter to iterate over all open forks. You can limit this iteration to a specific volume with the `volume` parameter.

volume

The volume to search, when iterating over multiple forks. To iterate over all open forks on a single volume, specify the volume reference number in this parameter. To iterate over all open forks on all volumes, set this parameter to the constant `kFSInvalidVolumeRefNum`.

This parameter is ignored if you specify a fork reference number in the `desiredRefNum` parameter. Set `desiredRefNum` to zero if you wish to iterate over multiple forks.

See `FSVolumeRefNum` (page 847) for a description of the `FSVolumeRefNum` data type.

iterator

A pointer to an iterator. If the `desiredRefNum` parameter is 0, the iterator maintains state between calls to `FSGetForkCBInfo`. Set the `iterator` parameter to 0 before you begin iterating, on the first call to `FSGetForkCBInfo`. On return, the iterator will be updated; pass this updated iterator in the `iterator` parameter of the next call to `FSIterateForks` to continue iterating.

actualRefNum

On return, a pointer to the reference number of the open fork. This parameter is optional if you do not wish to retrieve the fork's reference number, pass `NULL`.

forkInfo

On return, a pointer to an `FSForkInfo` structure containing information about the open fork. This parameter is optional; if you do not wish this information returned, set `forkInfo` to `NULL`. See `FSForkInfo` (page 832) for a description of the `FSForkInfo` data type.

On OS X, the value returned by `FSGetForkCBInfo` in the `physicalEOF` field of the `FSForkInfo` structure may differ from the physical file length reported by `FSGetCatalogInfo`, `PBGetCatInfo`, and related functions. When a write causes a file to grow in size, the physical length reported by `FSGetCatalogInfo` and similar calls increases by the clump size, which is a multiple of the allocation block size. However, the physical length returned by `FSGetForkCBInfo` changes according to the allocation block size and the file lengths returned by the respective functions get out of sync.

ref

On return, a pointer to the `FSRef` for the file or directory that contains the fork. This parameter is optional; if you do not wish to retrieve the `FSRef`, set `ref` to `NULL`. See `FSRef` (page 837) for a description of the `FSRef` data type.

outForkName

On return, a pointer to the name of the fork. This parameter is optional; if you do not wish the name returned, set `outForkName` to `NULL`. See `HFSUniStr255` (page 855) for a description of the `HFSUniStr255` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943). If you are iterating over multiple forks, the function returns `errFSNoMoreItems` if there are no more open forks to return.

Discussion

Carbon applications are no longer guaranteed access to the FCB table. Instead, applications should use `FSGetForkCBInfo`, or one of the related parameter block functions, `PBGetForkCBInfoSync` (page 661) and `PBGetForkCBInfoAsync` (page 660) , to access information about a fork control block.

Special Considerations

Returning the fork information in the `forkInfo` parameter generally does not require a disk access; returning the information in the `ref` or `forkName` parameters may cause disk access for some volume formats.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSGetForkPosition

Returns the current position of an open fork.

```
OSErr FSGetForkPosition (
   FSIORefNum forkRefNum,
   SInt64 *position
);
```

Parameters

forkRefNum

The reference number of a fork previously opened by the `FSOpenFork` (page 514) function or one of its corresponding parameter block calls, `PBOpenForkSync` (page 740) and `PBOpenForkAsync` (page 739).

position

On return, a pointer to the current position of the fork. The returned fork position is relative to the start of the fork (that is, it is an absolute offset in bytes).

Return Value
A result code. See "File Manager Result Codes" (page 943).

Special Considerations

Before calling the `FSGetForkPosition` function, call the `Gestalt` function with the `gestaltFSAttr` selector to determine if `FSGetForkPosition` is available. If the function is available, but is not directly supported by a volume, the File Manager will automatically call `PBGetFPosSync` (page 666); however, you will not be able to determine the fork position of a named fork other than the data or resource fork, or of a fork larger than 2 GB.

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

Declared In
`Files.h`

FSGetForkSize

Returns the size of an open fork.

```
OSErr FSGetForkSize (
   FSIORefNum forkRefNum,
   SInt64 *forkSize
);
```

Parameters

forkRefNum

The reference number of the open fork. You can obtain this fork reference number with the `FSOpenFork` (page 514) function, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 740) and `PBOpenForkAsync` (page 739).

forkSize

On return, a pointer to the logical size (the logical end-of-file) of the fork, in bytes. The size returned is the total number of bytes that can be read from the fork; the amount of space actually allocated on the volume (the physical size) will probably be larger.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Special Considerations

To determine whether the `FSGetForkSize` function is present, call the `Gestalt` function. If `FSGetForkSize` is present, but is not directly supported by a volume, the File Manager will call `PBGetEOFSync` (page 655); however, you will not be able to determine the size of a fork other than the data or resource fork, or of a fork larger than 2 GB.

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

Declared In
`Files.h`

FSGetResourceForkName

Returns a Unicode string constant for the name of the resource fork.

```
OSErr FSGetResourceForkName (
   HFSUniStr255 *resourceForkName
);
```

Parameters

resourceForkName

On input, a pointer to an `HFSUniStr255` structure. On return, this structure contains the Unicode name of the resource fork. Currently, this is "RESOURCE_FORK". See `HFSUniStr255` (page 855) for a description of the `HFSUniStr255` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
There is no parameter block-based form of this call since it is not dispatched to individual volume formats, and does not require any I/O.

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

Declared In
`Files.h`

FSGetVolumeInfo

Returns information about a volume.

```
OSErr FSGetVolumeInfo (
   FSVolumeRefNum volume,
   ItemCount volumeIndex,
   FSVolumeRefNum *actualVolume,
   FSVolumeInfoBitmap whichInfo,
   FSVolumeInfo *info,
   HFSUniStr255 *volumeName,
   FSRef *rootDirectory
);
```

Parameters

volume

If you wish to obtain information on a particular volume, pass that volume's reference number here. If you wish to index through the list of mounted volumes, pass the constant `kFSInvalidVolumeRefNum` in this parameter. See `FSVolumeRefNum` (page 847) for a description of the `FSVolumeRefNum` data type.

volumeIndex

The index of the desired volume, or 0 to use the volume reference number in the `volume` parameter.

actualVolume

On return, a pointer to the volume reference number of the volume. This is useful when indexing over all mounted volumes. If you don't want this information (if, for instance, you supplied a particular volume reference number in the `volume`) parameter, set `actualVolume` to `NULL`.

whichInfo

A bitmap specifying which volume information fields to get and return in the `info` parameter. If you don't want information about the volume returned in the `info` parameter, set `whichInfo` to `kFSVolInfoNone`. See "Volume Information Bitmap Constants" (page 938) for a description of the bits in this parameter.

info

On return, a pointer to the volume information. If you don't want this output, set this parameter to `NULL`. See `FSVolumeInfo` (page 842) for a description of the `FSVolumeInfo` data type.

volumeName

On return, a pointer to the Unicode name of the volume. If you do not wish the name returned, pass `NULL`. See `HFSUniStr255` (page 855) for a description of the `HFSUniStr255` data type.

rootDirectory

On return, a pointer to the `FSRef` for the volume's root directory. If you do not wish the root directory returned, pass `NULL`. See `FSRef` (page 837) for a description of the `FSRef` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

You can specify a particular volume or index through the list of mounted volumes. To get information on a particular volume, pass the volume reference number of the desired volume in the `volume` parameter and set the `volumeIndex` parameter to zero. To index through the list of mounted volumes, pass `kFSInvalidVolumeRefNum` in the volume parameter and set `volumeIndex` to the index, starting at 1 with the first call to `FSGetVolumeInfo`.

When indexing through the list of mounted volumes, you may encounter an error with a particular volume. The terminating error code for full traversal of this list is `nsvErr`. In order to completely traverse the entire list, you may have to bump the index count when encountering other errors (for example, `ioErr`).

To get information about the root directory of a volume, use the `FSGetCatalogInfo` (page 494) function, or one of the corresponding parameter block calls, `PBGetCatalogInfoSync` (page 647) and `PBGetCatalogInfoAsync` (page 643).

Special Considerations

After an operation that changes the amount of free space on the volume—such as deleting a file—there may be a delay before a call to `FSGetVolumeInfo` returns the updated amount. This is because the File Manager caches and periodically updates file system information, to reduce the number of calls made to retrieve the information from the file system. Currently, the File Manager updates its information every 15 seconds. This primarily affects NFS volumes. DOS, SMB, UFS and WebDAV volumes were also affected by this in previous versions of Mac OS X, but behave correctly in Mac OS X version 10.3 and later.

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

Declared In
`Files.h`

FSGetVolumeMountInfo

Retrieves the mounting information associated with the specified volume.

```
OSStatus FSGetVolumeMountInfo (
   FSVolumeRefNum volume,
   BytePtr buffer,
   ByteCount bufferSize,
   ByteCount *actualSize
);
```

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

Declared In
`Files.h`

FSGetVolumeMountInfoSize

Determines the size of the mounting information associated with the specified volume.

```
OSStatus FSGetVolumeMountInfoSize (
   FSVolumeRefNum volume,
   ByteCount *size
);
```

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

Declared In
`Files.h`

FSGetVolumeParms

Retrieves information about the characteristics of a volume.

```
OSStatus FSGetVolumeParms (
   FSVolumeRefNum volume,
   GetVolParmsInfoBuffer *buffer,
   ByteCount bufferSize
);
```

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

Declared In
`Files.h`

FSIterateForks

Determines the name and size of every named fork belonging to a file or directory.

```
OSErr FSIterateForks (
   const FSRef *ref,
   CatPositionRec *forkIterator,
   HFSUniStr255 *forkName,
   SInt64 *forkSize,
   UInt64 *forkPhysicalSize
);
```

Parameters

ref

A pointer to an `FSRef` specifying the file or directory to iterate. See `FSRef` (page 837) for a description of the `FSRef` data type.

forkIterator

A pointer to a structure which maintains state between calls to `FSIterateForks`. Before the first call, set the `initialize` field of the structure to 0. The fork iterator will be updated after the call completes; the updated iterator should be passed into the next call. See `CatPositionRec` (page 801) for a description of the `CatPositionRec` data type.

forkName

On return, a pointer to the Unicode name of the fork. This parameter is optional; if you do not wish the name returned, pass a `NULL` pointer. See `HFSUniStr255` (page 855) for a description of the `HFSUniStr255` data type.

forkSize

On return, a pointer to the logical size of the fork, in bytes. This parameter is optional; if you do not wish to retrieve the logical fork size, pass a `NULL` pointer.

forkPhysicalSize

On return, a pointer to the physical size of the fork (that is, to the amount of space allocated on disk), in bytes. This parameter is optional; if you do not wish to retrieve the physical fork size, pass a `NULL` pointer.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
Since information is returned about one fork at a time, several calls may be required to iterate through all the forks. There is no guarantee about the order in which forks are returned; the order may vary between iterations.

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

Declared In
`Files.h`

FSLockRange

Locks a range of bytes of the specified fork.

```
OSStatus FSLockRange (
    FSIORefNum forkRefNum,
    UInt16 positionMode,
    SInt64 positionOffset,
    UInt64 requestCount,
    UInt64 *rangeStart
);
```

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

Declared In
`Files.h`

FSMakeFSRefUnicode

Constructs an `FSRef` for a file or directory, given a parent directory and a Unicode name.

```
OSErr FSMakeFSRefUnicode (
   const FSRef *parentRef,
   UniCharCount nameLength,
   const UniChar *name,
   TextEncoding textEncodingHint,
   FSRef *newRef
);
```

Parameters

parentRef

A pointer to the `FSRef` of the parent directory of the file or directory for which to create a new `FSRef`. See `FSRef` (page 837) for a description of the `FSRef` data type.

nameLength

The length of the file or directory name.

name

A pointer to the Unicode name for the file or directory. The name must be a leaf name; partial or full pathnames are not allowed. If you have a partial or full pathname in Unicode, you will have to parse it yourself and make multiple calls to `FSMakeFSRefUnicode`.

textEncodingHint

The suggested text encoding to use when converting the Unicode name of the file or directory to some other encoding. If you pass the constant `kTextEncodingUnknown`, the File Manager will use a default value.

newRef

On return, if the function returns a result of `noErr`, a pointer to the new `FSRef`.

Return Value
A result code. See "File Manager Result Codes" (page 943).

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

Related Sample Code
BSDLLCTest
CarbonSketch
QTCarbonShell

Declared In
`Files.h`

FSMakeFSSpec

Creates an `FSSpec` structure describing a file or directory. (Deprecated in Mac OS X v10.4. Use `FSMakeFSRefUnicode` (page 504) instead.)

```
OSErr FSMakeFSSpec (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName,
   FSSpec *spec
);
```

Parameters

vRefNum

A volume specification for the volume containing the file or directory. This parameter can contain a volume reference number, a drive number, or 0 to specify the default volume.

dirID

The parent directory ID of the target object. If the directory is sufficiently specified in the `fileName` parameter, the `dirID` parameter can be set to 0. If the `fileName` parameter contains an empty string, `FSMakeFSSpec` creates an `FSSpec` structure for the directory specified by the `dirID` parameter.

fileName

A full or partial pathname. If the `fileName` parameter specifies a full pathname, `FSMakeFSSpec` ignores both the `vRefNum` and `dirID` parameters. A partial pathname might identify only the final target, or it might include one or more parent directory names. If `fileName` specifies a partial pathname, then `vRefNum`, `dirID`, or both must be valid.

spec

A pointer to a file system specification to be filled in by `FSMakeFSSpec`. The `FSMakeFSSpec` function fills in the fields of the file system specification using the information contained in the other three parameters. If your application receives any result code other than `noErr` or `fnfErr`, all fields of the resulting `FSSpec` structure are set to 0.

The file system specification structure that you pass in this parameter should not share storage space with the input pathname; the `name` field may be initialized to the empty string before the pathname has been processed. For example, `fileName` should not refer to the `name` field of the output file system specification.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
You should call `FSMakeFSSpec`, or one of the corresponding parameter block functions, `PBMakeFSSpecSync` (page 736) and `PBMakeFSSpecAsync` (page 734) , whenever you want to create an `FSSpec` structure. You should not create an `FSSpec` by filling in the fields of the structure yourself.

If the specified volume is mounted and the specified parent directory exists, but the target file or directory doesn't exist in that location, `FSMakeFSSpec` fills in the structure and then returns `fnfErr` instead of `noErr`. The structure is valid, but it describes a target that doesn't exist. You can use the structure for other operations, such as creating a file with the `FSpCreate` (page 525) function.

Carbon Porting Notes

Non-Carbon applications can also specify a working directory reference number in the `vRefNum` parameter. However, because working directories are not supported in Carbon, you cannot specify a working directory reference number if you wish your application to be Carbon-compatible.

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
QTMetaData

Simple DrawSprocket

Declared In
`Files.h`

FSMountLocalVolumeAsync

Mounts a volume asynchronously.

```
OSStatus FSMountLocalVolumeAsync (
   CFStringRef diskID,
   CFURLRef mountDir,
   FSVolumeOperation volumeOp,
   void *clientData,
   OptionBits flags,
   FSVolumeMountUPP callback,
   CFRunLoopRef runloop,
   CFStringRef runloopMode
);
```

Parameters

diskID

The disk to mount.

mountDir

Pass in `NULL` ; currently only `NULL` is supported.

volumeOp

An `FSVolumeOperation` returned by `FSCreateVolumeOperation`

clientData

A pointer to client data associated with the operation. This parameter can be `NULL`.

flags

Options for future use.

callback

The function to call when mount is complete. This parameter can be `NULL`.

runloop

The runloop to run on.

runloopMode

The mode for the runloop.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

This function starts the process to mount the disk specified by the *diskID* parameter at the location specified by the *mountDir* parameter. If *mountDir* is `NULL`, the default location is used. If a callback function is provided, that function will be called when the mount operation is complete. Once this function returns `noErr` the status of the operation can be found using the `FSGetAsyncMountStatus` function.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Files.h`

FSMountLocalVolumeSync

Mounts a volume.

```
OSStatus FSMountLocalVolumeSync (
   CFStringRef diskID,
   CFURLRef mountDir,
   FSVolumeRefNum *mountedVolumeRefNum,
   OptionBits flags
);
```

Parameters

diskID

The disk to mount.

mountDir

Pass in `NULL`; currently only `NULL` is supported.

mountedVolumeRefNum

On return, a pointer to the volume reference number of the newly mounted volume.

flags

Options for future use.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

This function mounts the disk specified by the *diskID* parameter at the location specified by the *mountDir* parameter. If *mountDir* is `NULL`, the default location is used. This function returns after the mount is complete.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Files.h`

FSMountServerVolumeAsync

Mounts a server volume asynchronously.

```
OSStatus FSMountServerVolumeAsync (
   CFURLRef url,
   CFURLRef mountDir,
   CFStringRef user,
   CFStringRef password,
   FSVolumeOperation volumeOp,
   void *clientData,
   OptionBits flags,
   FSVolumeMountUPP callback,
   CFRunLoopRef runloop,
   CFStringRef runloopMode
);
```

Parameters

url

The server to mount.

mountDir

The directory to mount the server to. If this parameter is `NULL`, the default location is used.

user
A string to pass as the user for authentication. This parameter can be NULL.

password
A string to pass as the password for authenticated log in. This parameter can be NULL.

volumeOp
An FSVolumeOperation returned by the FSCreateVolumeOperation function.

clientData
A pointer to client data associated with the operation. This parameter can be NULL.

flags
Options for future use.

callback
A function to call when the mount is complete. This parameter can be NULL.

runloop
The runloop to run on.

runloopMode
The mode for the runloop.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
This function will start the process to mount the server specified by the *url* parameter at the location specified by the *mountDir* parameter. If *mountDir* is NULL, the default location is used. An optional user and password can be passed in for authentication. If no user or password is provided then the underlying file system will handle authentication if required. If a callback function is provided, that function will be called when the mount operation is complete. Once this function returns noErr the status of the operation can be found using the FSGetAsyncMountStatus function.

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

Declared In
Files.h

FSMountServerVolumeSync

Mounts a server volume.

```
OSStatus FSMountServerVolumeSync (
   CFURLRef url,
   CFURLRef mountDir,
   CFStringRef user,
   CFStringRef password,
   FSVolumeRefNum *mountedVolumeRefNum,
   OptionBits flags
);
```

Parameters

url
The server to mount.

mountDir
The directory to mount the server to. If this parameter is NULL, the default location is used.

user
A string to pass as the user for authentication.

password
A string to pass as the password for authenticated log in.

mountedVolumeRefNum
On return, a pointer to the volume reference number of the newly mounted volume.

flags
Options for future use.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
This function will mount the server specified by the *url* parameter at the location specified by the *mountDir* parameter. If *mountDir* is NULL, the default location is used. An optional user and password can be passed in for authentication. If no user or password is provided then the underlying file system will handle authentication if required. This function returns after the mount is complete.

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

Declared In
Files.h

FSMoveObject

Moves a file or directory into a different directory.

```
OSErr FSMoveObject (
   const FSRef *ref,
   const FSRef *destDirectory,
   FSRef *newRef
);
```

Parameters

ref
A pointer to an FSRef specifying the file or directory to move. See FSRef (page 837) for a description of the FSRef data type.

destDirectory
A pointer to an FSRef specifying the directory into which the file or directory indicated by the ref parameter will be moved.

newRef
On return, a pointer to the new FSRef for the file or directory in its new location. This parameter is optional; if you do not wish the FSRef returned, pass NULL.

Return Value
A result code. See "File Manager Result Codes" (page 943). If the destDirectory parameter specifies a non-existent object, dirNFErr is returned; if it refers to a file, errFSNotAFolder is returned. If the directory specified in the destDirectory parameter is on a different volume than the file or directory indicated in the ref parameter, diffVolErr is returned.

Discussion
Moving an object may change its `FSRef`. If you want to continue to refer to the object, you should pass a non- `NULL` pointer in the `newRef` parameter and use the `FSRef` returned there to refer to the object after the move. The original `FSRef` passed in the `ref` parameter may or may not be usable after the move. The `newRef` parameter may point to the same storage as the `destDirectory` or `ref` parameters.

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

Declared In
`Files.h`

FSMoveObjectAsync

Starts an asynchronous file operation to move a source object to a destination directory.

```
OSStatus FSMoveObjectAsync (
   FSFileOperationRef fileOp,
   const FSRef *source,
   const FSRef *destDir,
   CFStringRef destName,
   OptionBits flags,
   FSFileOperationStatusProcPtr callback,
   CFTimeInterval statusChangeInterval,
   FSFileOperationClientContext *clientContext
);
```

Parameters

`fileOp`
The file operation object you created for this move operation.

`source`
A pointer to the source object to move. The object can be a file or a directory.

`destDir`
A pointer to the destination directory. If the destination directory is not on the same volume as the source object, the source object is copied and then deleted.

`destName`
The name for the new object in the destination directory. Pass `NULL` to use the name of the source object.

`flags`
One or more file operation option flags. See "File Operation Options" (page 917). If you specify the `kFSFileOperationDoNotMoveAcrossVolumes` flag and the destination directory is not on the same volume as the source object, this function does nothing and returns an error.

`callback`
A callback function to receive status updates as the file operation proceeds. For more information, see "File Operation Callbacks" (page 788). This parameter is optional; pass `NULL` if you don't need to supply a status callback.

`statusChangeInterval`
The minimum time in seconds between callbacks within a single stage of an operation.

clientContext

User-defined data to associate with this operation. For more information, see `FSFileOperationClientContext` (page 829). This parameter is optional; pass `NULL` if you don't need to supply a client context.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
If you specify a status callback function, status callbacks will occur in one of the run loop and mode combinations with which you scheduled the file operation.

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

Declared In
`Files.h`

FSMoveObjectSync

Moves a source object to a destination directory.

```
OSStatus FSMoveObjectSync (
   const FSRef *source,
   const FSRef *destDir,
   CFStringRef destName,
   FSRef *target,
   OptionBits options
);
```

Parameters

source

A pointer to the source object to move. The object can be a file or a directory. On output, the source object is no longer valid; if you want to refer to the moved object, you should use the `FSRef` variable passed back in the `target` parameter.

destDir

A pointer to the destination directory. If the destination directory is not on the same volume as the source object, the source object is copied and then deleted.

destName

The name for the new object in the destination directory. Pass `NULL` to use the name of the source object.

target

A pointer to an `FSRef` variable that, on output, refers to the new object in the destination directory. This parameter is optional; pass `NULL` if you don't need to refer to the new object.

options

One or more file operation option flags. See "File Operation Options" (page 917). If you specify the `kFSFileOperationDoNotMoveAcrossVolumes` flag and the destination directory is not on the same volume as the source object, this function does nothing and returns an error.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
If the destination directory is on the same volume as the source object, this is a fast operation. If the move is across volumes, this function could take a significant amount of time to execute; you should either call it in a thread other than the main thread or use `FSMoveObjectAsync` (page 511) instead.

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

Declared In
`Files.h`

FSMoveObjectToTrashAsync

Starts an asynchronous file operation to move a source object to the Trash.

```
OSStatus FSMoveObjectToTrashAsync (
   FSFileOperationRef fileOp,
   const FSRef *source,
   OptionBits flags,
   FSFileOperationStatusProcPtr callback,
   CFTimeInterval statusChangeInterval,
   FSFileOperationClientContext *clientContext
);
```

Parameters

fileOp
The file operation object you created for this move operation. For more information, see the function `FSFileOperationCreate` (page 488).

source
A pointer to the source object to move. The object can be a file or a directory.

flags
One or more file operation option flags. See "File Operation Options" (page 917).

callback
A callback function to receive status updates as the file operation proceeds. For more information, see "File Operation Callbacks" (page 788). This parameter is optional; pass `NULL` if you don't need to supply a status callback.

statusChangeInterval
The minimum time in seconds between callbacks within a single stage of an operation.

clientContext
User-defined data to associate with this operation. This data is passed to the function you specify in the *callback* parameter. For more information, see `FSFileOperationClientContext` (page 829). This parameter is optional; pass `NULL` if you don't need to supply a client context.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
This function starts an asynchronous file operation to move the object specified by the *source* parameter to the Trash. If the source volume does not support a trash folder, the operation will fail and return an error to the status callback specified in the *callback* parameter. (This is the same circumstance that triggers the delete immediately behavior in the Finder.)

Status callbacks occur on one of the runloop and mode combinations on which the operation was scheduled. Upon successful completion of the operation, the last *currentItem* parameter (passed to the last status callback or retrieved by calling `FSFileOperationCopyStatus` (page 487)) is the object in the Trash.

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

Declared In
`Files.h`

FSMoveObjectToTrashSync

Moves a source object to the Trash.

```
OSStatus FSMoveObjectToTrashSync (
   const FSRef *source,
   FSRef *target,
   OptionBits options
);
```

Parameters

source
: A pointer to the source object to move. The object can be a file or a directory. On output, the source object is no longer valid; if you want to refer to the moved object, you should use the value passed back in the *target* parameter.

target
: A pointer to the target object that, on output, resides in a trash folder. This parameter is optional; pass `NULL` if you don't need to refer to this object.

options
: One or more file operation option flags. See "File Operation Options" (page 917).

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
This function moves a file or directory to the Trash, adjusting the object's name if necessary. The appropriate trash folder is chosen based on the source volume and the current user. If the source volume does not support a trash folder, this function does nothing and returns an error. (This is the same circumstance that triggers the delete immediately behavior in the Finder.)

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

Declared In
`Files.h`

FSOpenFork

Opens any fork of a file or directory for streaming access.

```
OSErr FSOpenFork (
   const FSRef *ref,
   UniCharCount forkNameLength,
   const UniChar *forkName,
   SInt8 permissions,
   FSIORefNum *forkRefNum
);
```

Parameters

ref

A pointer to an `FSRef` specifying the file or directory owning the fork to open. See `FSRef` (page 837) for a description of the `FSRef` data type.

forkNameLength

The length of the fork name in Unicode characters.

forkName

A pointer to the Unicode name of the fork to open. You can obtain the string constants for the data fork and resource fork names using the `FSGetDataForkName` (page 497) and `FSGetResourceForkName` (page 500) functions. All volume formats should support data and resource forks; other named forks may be supported by some volume formats.

permissions

A constant indicating the type of access which you wish to have to the fork via the returned fork reference. This parameter is the same as the `permission` parameter passed to the `FSpOpenDF` and `FSpOpenRF` functions. For a description of the types of access which you can request, see "File Access Permission Constants" (page 908).

forkRefNum

On return, a pointer to the fork reference number for accessing the open fork.

Return Value

A result code. See "File Manager Result Codes" (page 943). On some file systems, `FSOpenFork` will return the error `eofErr` if you try to open the resource fork of a file for which no resource fork exists with read-only access.

Discussion

When you use this function to open a file on a local volume and pass in a permissions value of `fsCurPerm`, `fsWrPerm`, or `fsRdWrPerm`, Mac OS X does not guarantee exclusive file access. Before making any assumptions about the underlying file access, you should always check to see whether the Supports Exclusive Locks feature is available. If this feature is not available, your application cannot know whether another application has access to the same file. For more information, see ADC Technical Note TN2037.

To access named forks or forks larger than 2GB, you must use the `FSOpenFork` function or one of the corresponding parameter block calls: `PBOpenForkSync` and `PBOpenForkAsync`. To determine if the `FSOpenFork` function is present, call the `Gestalt` function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

FSOpenIterator

Creates a catalog iterator that can be used to iterate over the contents of a directory or volume.

```
OSErr FSOpenIterator (
   const FSRef *container,
   FSIteratorFlags iteratorFlags,
   FSIterator *iterator
);
```

Parameters

container

A pointer to an FSRef for the directory to iterate. The set of items to iterate over can either be the objects directly contained in the directory, or all items directly or indirectly contained in the directory (in which case, the specified directory is the root of the subtree to iterate). See FSRef (page 837) for a description of the FSRef data type.

iteratorFlags

A set of flags which controls whether the iterator iterates over subtrees or just the immediate children of the container. See "Iterator Flags" (page 924) for a description of the flags defined for this parameter.

Iteration over subtrees which do not originate at the root directory of a volume are not currently supported, and passing the kFSIterateSubtree flag in this parameter returns errFSBadIteratorFlags. To determine whether subtree iterators are supported, check that the bSupportsSubtreeIterators bit returned by PBHGetVolParmsSync (page 695) or PBHGetVolParmsAsync (page 694) is set.

iterator

On return, a pointer to the new FSIterator. You can pass this iterator to the FSGetCatalogInfoBulk (page 495) or FSCatalogSearch (page 472) functions and their parameter block-based counterparts.

The iterator is automatically initialized so that the next use of the iterator returns the first item. The order that items are returned in is volume format dependent and may be different for two different iterators created with the same container and flags.

See FSIterator (page 835) for a description of the FSIterator data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
Catalog iterators must be closed when you are done using them, whether or not you have iterated over all the items. Iterators are automatically closed upon process termination, just like open files. However, you should use the FSCloseIterator (page 475) function, or one of the related parameter block functions, PBCloseIteratorSync (page 584) and PBCloseIteratorAsync (page 584) , to close an iterator to free up any system resources allocated to the iterator.

Before calling this function, you should check that it is present, by calling the Gestalt function.

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

Related Sample Code
QTCarbonShell

Declared In
Files.h

FSPathCopyObjectAsync

Starts an asynchronous file operation to copy a source object to a destination directory using pathnames.

```
OSStatus FSPathCopyObjectAsync (
   FSFileOperationRef fileOp,
   const char *sourcePath,
   const char *destDirPath,
   CFStringRef destName,
   OptionBits flags,
   FSPathFileOperationStatusProcPtr callback,
   CFTimeInterval statusChangeInterval,
   FSFileOperationClientContext *clientContext
);
```

Parameters

fileOp
: The file operation object you created for this copy operation.

sourcePath
: The UTF-8 pathname of the source object to copy. The object can be a file or a directory.

destDirPath
: The UTF-8 pathname of the destination directory.

destName
: The name for the new object in the destination directory. Pass `NULL` to use the name of the source object.

flags
: One or more file operation option flags. See "File Operation Options" (page 917).

callback
: A callback function to receive status updates as the file operation proceeds. For more information, see "File Operation Callbacks" (page 788). This parameter is optional; pass `NULL` if you don't need to supply a status callback.

statusChangeInterval
: The minimum time in seconds between callbacks within a single stage of an operation.

clientContext
: User-defined data to associate with this operation. For more information, see `FSFileOperationClientContext` (page 829). This parameter is optional; pass `NULL` if you don't need to supply a client context.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

If you specify a status callback function, status callbacks will occur in one of the run loop and mode combinations with which you scheduled the file operation.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`Files.h`

FSPathCopyObjectSync

Copies a source object to a destination directory using pathnames.

```
OSStatus FSPathCopyObjectSync (
   const char *sourcePath,
   const char *destDirPath,
   CFStringRef destName,
   char **targetPath,
   OptionBits options
);
```

Parameters

sourcePath

The UTF-8 pathname of the source object to copy. The object can be a file or a directory.

destDirPath

The UTF-8 pathname of the destination directory.

destName

The name for the new object in the destination directory. Pass `NULL` to use the name of the source object.

targetPath

A pointer to a char* variable that, on output, refers to the UTF-8 pathname of the new object in the destination directory. If the operation fails, the pathname is set to `NULL`. When you no longer need the pathname, you should free it. This parameter is optional; pass `NULL` if you don't need the pathname.

options

One or more file operation option flags. See "File Operation Options" (page 917).

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

This function could take a significant amount of time to execute. To avoid blocking your user interface, you should either call this function in a thread other than the main thread or use `FSPathCopyObjectAsync` (page 517) instead.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`Files.h`

FSPathFileOperationCopyStatus

Gets a copy of the current status information for an asynchronous file operation that uses pathnames.

```
OSStatus FSPathFileOperationCopyStatus (
   FSFileOperationRef fileOp,
   char **currentItem,
   FSFileOperationStage *stage,
   OSStatus *error,
   CFDictionaryRef *statusDictionary,
   void **info
);
```

Parameters

fileOp

The file operation to access.

currentItem

A pointer to a char* variable. On output, the variable refers to the UTF-8 pathname of the object currently being moved or copied. If the operation is complete, this parameter refers to the target (the new object corresponding to the source object in the destination directory). You should free the pathname when you are finished using it.

stage

A pointer to a file operation stage variable. On output, the variable contains the current stage of the file operation.

error

A pointer to an error status variable. On output, the variable contains the current error status of the file operation.

statusDictionary

A pointer to a dictionary variable. On output, the variable contains a dictionary with more detailed status information. For information about the contents of the dictionary, see "File Operation Status Dictionary Keys" (page 919). You should release the dictionary when you are finished using it.

info

A pointer to a generic pointer. On output, the generic pointer refers to user-defined data associated with this file operation.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Availability

Available in Mac OS X v10.4 and later.

Declared In

Files.h

FSPathMakeRef

Converts a POSIX-style pathname into an `FSRef` structure.

```
OSStatus FSPathMakeRef (
    const UInt8 *path,
    FSRef *ref,
    Boolean *isDirectory
);
```

Parameters

path

A UTF-8 C string that contains the pathname to convert.

ref

A pointer to an `FSRef` structure allocated by the caller. On output, the `FSRef` structure refers to the object whose location is specified by the *path* parameter.

isDirectory

A pointer to a Boolean variable allocated by the caller. On output, `true` indicates the object is a directory. This parameter is optional and may be `NULL`.

Return Value
A result code. See "File Manager Result Codes" (page 943).

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

Related Sample Code
CocoaDVDPlayer

Declared In
`Files.h`

FSPathMakeRefWithOptions

Converts a POSIX-style pathname into an `FSRef` structure with options.

```
OSStatus FSPathMakeRefWithOptions (
   const UInt8 *path,
   OptionBits options,
   FSRef *ref,
   Boolean *isDirectory
);
```

Parameters

path

A UTF-8 C string that contains the pathname to convert.

options

One or more conversion flags. See "Path Conversion Options" (page 928).

ref

A pointer to an `FSRef` structure allocated by the caller. On output, the `FSRef` structure refers to the object whose location is specified by the *path* parameter. If the object is a symbolic link, the *options* parameter determines whether the `FSRef` structure refers to the link itself or to the linked object.

isDirectory

A pointer to a Boolean variable allocated by the caller. On output, `true` indicates the object is a directory. This parameter is optional and may be `NULL`.

Return Value
A result code. See "File Manager Result Codes" (page 943).

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

Declared In
`Files.h`

FSPathMoveObjectAsync

Starts an asynchronous file operation to move a source object to a destination directory using pathnames.

```
OSStatus FSPathMoveObjectAsync (
   FSFileOperationRef fileOp,
   const char *sourcePath,
   const char *destDirPath,
   CFStringRef destName,
   OptionBits flags,
   FSPathFileOperationStatusProcPtr callback,
   CFTimeInterval statusChangeInterval,
   FSFileOperationClientContext *clientContext
);
```

Parameters

fileOp
The file operation object you created for this move operation.

sourcePath
The UTF-8 pathname of the source object to move. The object can be a file or a directory.

destDirPath
The UTF-8 pathname of the destination directory. If the destination directory is not on the same volume as the source object, the source object is copied and then deleted.

destName
The name for the new object in the destination directory. Pass `NULL` to use the name of the source object.

flags
One or more file operation option flags. See "File Operation Options" (page 917). If you specify the `kFSFileOperationDoNotMoveAcrossVolumes` flag and the destination directory is not on the same volume as the source object, this function does nothing and returns an error.

callback
A callback function to receive status updates as the file operation proceeds. For more information, see "File Operation Callbacks" (page 788). This parameter is optional; pass `NULL` if you don't need to supply a status callback.

statusChangeInterval
The minimum time in seconds between callbacks within a single stage of an operation.

clientContext
User-defined data to associate with this operation. For more information, see `FSFileOperationClientContext` (page 829). This parameter is optional; pass `NULL` if you don't need to supply a client context.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
If you specify a status callback function, status callbacks will occur in one of the run loop and mode combinations with which you scheduled the file operation.

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

Declared In
`Files.h`

FSPathMoveObjectSync

Moves a source object to a destination directory using pathnames.

```
OSStatus FSPathMoveObjectSync (
   const char *sourcePath,
   const char *destDirPath,
   CFStringRef destName,
   char **targetPath,
   OptionBits options
);
```

Parameters

sourcePath
The UTF-8 pathname of the source object to move. The object can be a file or a directory.

destDirPath
The UTF-8 pathname of the destination directory. If the destination directory is not on the same volume as the source object, the source object is copied and then deleted.

destName
The name for the new object in the destination directory. Pass `NULL` to use the name of the source object.

targetPath
A pointer to a char* variable that, on output, refers to the UTF-8 pathname of the new object in the destination directory. When you no longer need the pathname, you should free it. If the operation fails, the pathname is set to `NULL`. This parameter is optional; pass `NULL` if you don't need the pathname.

options
One or more file operation option flags. See "File Operation Options" (page 917). If you specify the `kFSFileOperationDoNotMoveAcrossVolumes` flag and the destination directory is not on the same volume as the source object, this function does nothing and returns an error.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
If the destination directory is on the same volume as the source object, this is a fast operation. If the move is across volumes, this function could take a significant amount of time to execute; you should call it in a thread other than the main thread or use `FSPathMoveObjectAsync` (page 521) instead.

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

Declared In
`Files.h`

FSPathMoveObjectToTrashAsync

Starts an asynchronous file operation to move a source object, specified using a pathname, to the Trash.

```
OSStatus FSPathMoveObjectToTrashAsync (
   FSFileOperationRef fileOp,
   const char *sourcePath,
   OptionBits flags,
   FSPathFileOperationStatusProcPtr callback,
   CFTimeInterval statusChangeInterval,
   FSFileOperationClientContext *clientContext
);
```

Parameters

fileOp
The file operation object you created for this move operation. For more information, see the function `FSFileOperationCreate` (page 488).

sourcePath
The UTF-8 pathname of the source object to move. The object can be a file or a directory.

flags
One or more file operation option flags. See "File Operation Options" (page 917).

callback
A callback function to receive status updates as the file operation proceeds. For more information, see "File Operation Callbacks" (page 788). This parameter is optional; pass `NULL` if you don't need to supply a status callback.

statusChangeInterval
The minimum time in seconds between callbacks within a single stage of an operation.

clientContext
User-defined data to associate with this operation. This data is passed to the function you specify in the *callback* parameter. For more information, see `FSFileOperationClientContext` (page 829). This parameter is optional; pass `NULL` if you don't need to supply a client context.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
This function starts an asynchronous file operation to move the object specified by the *sourcePath* parameter to the Trash. If the source volume does not support a trash folder, the operation will fail and return an error to the status callback specified in the *callback* parameter. (This is the same circumstance that triggers the delete immediately behavior in the Finder.)

Status callbacks occur on one of the runloop and mode combinations on which the operation was scheduled. Upon successful completion of the operation, the last *currentItem* parameter (passed to the last status callback or retrieved by calling `FSFileOperationCopyStatus` (page 487)) is the object in the Trash.

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

Declared In
`Files.h`

FSPathMoveObjectToTrashSync

Moves a source object, specified using a pathname, to the Trash.

```
OSStatus FSPathMoveObjectToTrashSync (
   const char *sourcePath,
   char **targetPath,
   OptionBits options
);
```

Parameters

sourcePath
The UTF-8 pathname of the source object to move. The object can be a file or a directory.

targetPath
A pointer to a char* variable that, on output, refers to the UTF-8 pathname of the target object in the Trash. When you no longer need the pathname, you should free it. If the operation fails, the pathname is set to `NULL`. This parameter is optional; pass `NULL` if you don't need the pathname.

options
One or more file operation option flags. See "File Operation Options" (page 917).

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
This function moves a file or directory to the Trash, adjusting the object's name if necessary. The appropriate trash folder is chosen based on the source volume and the current user. If the source volume does not support a trash folder, this function does nothing and returns an error. (This is the same circumstance that triggers the delete immediately behavior in the Finder.)

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

Declared In
`Files.h`

FSpCatMove

Moves a file or directory from one location to another on the same volume. (Deprecated in Mac OS X v10.4. Use `FSMoveObject` (page 510) instead.)

```
OSErr FSpCatMove (
   const FSSpec *source,
   const FSSpec *dest
);
```

Parameters

source

A pointer to an `FSSpec` structure specifying the name and location of the file or directory to move. See `FSSpec` (page 840) for a description of the `FSSpec` data type.

dest

A pointer to an `FSSpec` structure specifying the name and location of the directory into which the source file or directory is to be moved. The directory ID specified in the `parID` field of this `FSSpec` is the directory ID of the parent of the directory into which you want to move the source file or directory. The `name` field of this `FSSpec` specifies the name of the directory into which you want to move the source file or directory.

If you don't already know the parent directory ID of the destination directory, it might be easier to use the `PBCatMoveSync` or `PBCatMoveAsync` function, which allow you to specify only the directory ID of the destination directory.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The `FSpCatMove` function is strictly a file catalog operation; it does not actually change the location of the file or directory on the disk. You cannot use `FSpCatMove` to move a file or directory to another volume (that is, the `vRefNum` field in both `FSSpec` structures in the `source` and `dest` parameters must be the same). Also, you cannot use `FSpCatMove` to rename files or directories; to rename a file or directory, use `FSpRename` (page 533).

If you need to move files or directories with named forks other than the data and resource forks, with long Unicode names, or files larger than 2GB, you should use the `FSMoveObject` (page 510) function, or one of the corresponding parameter block calls, `PBMoveObjectSync` (page 738) and `PBMoveObjectAsync` (page 737).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

FSpCreate

Creates a new file. (Deprecated in Mac OS X v10.4. Use `FSCreateFileUnicode` (page 481) instead.)

```
OSErr FSpCreate (
   const FSSpec *spec,
   OSType creator,
   OSType fileType,
   ScriptCode scriptTag
);
```

Parameters

spec

A pointer to an `FSSpec` structure specifying the file to be created. See `FSSpec` (page 840) for a description of the `FSSpec` data type.

creator

The creator of the new file. See the documentation for the Finder Interface for more information on file creators.

fileType

The file type of the new file. See the documentation for the Finder Interface for more information on file types.

scriptTag

The code of the script system in which the filename is to be displayed. If you have established the name and location of the new file using either the `NavAskSaveChanges` or `NavCustomAskSaveChanges` function, specify the script code returned in the reply structure. Otherwise, specify the system script by setting the `scriptTag` parameter to the value `smSystemScript`.

For more information about the functions `NavAskSaveChanges` and `NavCustomAskSaveChanges`, see *Programming With Navigation Services*. See the *Script Manager Reference* for a description of the `smSystemScript` constant.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The `FSpCreate` function creates a new file (both data and resource forks) with the specified type, creator, and script code. The new file is unlocked and empty. The date and time of creation and last modification are set to the current date and time.

Files created using `FSpCreate` are not automatically opened. If you want to write data to the new file, you must first open the file using one of the file access functions, `FSpOpenDF` (page 531) , `HOpenDF` (page 554) , `PBHOpenDFSync` (page 708) or `PBHOpenDFAsync` (page 706).

The resource fork of the new file exists but is empty. You'll need to call one of the Resource Manager functions `HCreateResFile` or `FSpCreateResFile` to create a resource map in the file before you can open it by calling one of the Resource Manager functions `HOpenResFile` or `FSpOpenResFile`).

Before calling this function, you should call the `Gestalt` function to check that the function is available. If `FSpCreate` is not available, you can use the function `HCreate` (page 550) instead. To create a file with a Unicode filename, use the function `FSCreateFileUnicode` (page 481) , or one of the corresponding parameter block calls, `PBCreateFileUnicodeSync` (page 593) and `PBCreateFileUnicodeAsync` (page 591).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Files.h`

FSpDelete

Deletes a file or directory. (Deprecated in Mac OS X v10.4. Use `FSDeleteObject` (page 484) instead.)

```
OSErr FSpDelete (
   const FSSpec *spec
);
```

Parameters

spec

A pointer to an `FSSpec` structure specifying the file or directory to delete. See `FSSpec` (page 840) for a description of the `FSSpec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943). If you attempt to delete an open file or a non-empty directory, `FSpDelete` returns the result code `fBsyErr`. `FSpDelete` also returns the result code `fBsyErr` if the directory has an open working directory associated with it.

Discussion
If the specified target is a file, both forks of the file are deleted. The file ID reference, if any, is removed. A file must be closed before you can delete it. Similarly, a directory must be empty before you can delete it.

Before calling this function, you should call the `Gestalt` function to check that the function is available. If `FSpDelete` is not available, you can use the function `HDelete` (page 551) instead.

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
CarbonSketch

Declared In
`Files.h`

FSpDirCreate

Creates a new directory. (Deprecated in Mac OS X v10.4. Use `FSCreateDirectoryUnicode` (page 479) instead.)

```
OSErr FSpDirCreate (
   const FSSpec *spec,
   ScriptCode scriptTag,
   SInt32 *createdDirID
);
```

Parameters

spec

A pointer to an `FSSpec` structure specifying the directory to be created.

Note that if the parent directory ID for the directory described by this `FSSpec` is 0 and the volume specified in this `FSSpec` is the default volume, the new directory is placed in the default directory of the volume. If the parent directory ID is 0 and the specified volume is a volume other than the default volume, the new directory is placed in the root directory of the volume. To create a directory at the root of a volume, regardless of whether that volume is the current default volume, set the parent directory ID to the constant `fsRtDirID`(2).

scriptTag

The code of the script system in which the directory name is to be displayed. If you have established the name and location of the new directory using either the `NavAskSaveChanges` or `NavCustomAskSaveChanges` function, specify the script code returned in the reply structure. Otherwise, specify the system script by setting the `scriptTag` parameter to the value `smSystemScript`.

For more information on the functions `NavAskSaveChanges` and `NavCustomAskSaveChanges`, see *Programming With Navigation Services*. For a description of the `smSystemScript` constant, see the *Script Manager Reference*.

createdDirID

On return, a pointer to the directory ID of the directory that was created.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The `FSpDirCreate` function sets the date and time of creation and last modification to the current date and time.

Before calling this function, you should call the `Gestalt` function to check that the function is available. If `FSpDirCreate` is not available, you can use the function `DirCreate` (page 463) instead. To create a directory with a Unicode name, use the function `FSCreateDirectoryUnicode` (page 479) , or one of the corresponding parameter block calls, `PBCreateDirectoryUnicodeSync` (page 589) and `PBCreateDirectoryUnicodeAsync` (page 587).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

FSpExchangeFiles

Exchanges the data stored in two files on the same volume. (Deprecated in Mac OS X v10.4. Use `FSExchangeObjects` (page 486) instead.)

```
OSErr FSpExchangeFiles (
   const FSSpec *source,
   const FSSpec *dest
);
```

Parameters

source

A pointer to an `FSSpec` for the first file to swap. The contents of this file and its file information are placed in the file specified in the `dest` parameter. See `FSSpec` (page 840) for a description of the `FSSpec` data type.

dest

A pointer to an `FSSpec` for the second file to swap. The contents of this file and its file information are placed in the file specified in the `source` parameter.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The `FSpExchangeFiles` function swaps the data in two files by changing the information in the volume's catalog and, if either of the files are open, in the file control blocks. The following fields in the catalog entries for the files are exchanged:

- `ioFlStBlk`
- `ioFlLgLen`
- `ioFlPyLen`
- `ioFlRStBlk`
- `ioFlRLgLen`
- `ioFlRPyLen`
- `ioFlMdDat`

In the file control blocks, the `fcbFlNum`, `fcbDirID`, and `fcbCName` fields are exchanged.

You should use `FSpExchangeFiles` when updating an existing file, so that the file ID remains valid in case the file is being tracked through its file ID. The `FSpExchangeFiles` function changes the fields in the catalog entries that record the location of the data and the modification dates. It swaps both the data forks and the resource forks.

The `FSpExchangeFiles` function works on both open and closed files. If either file is open, `FSpExchangeFiles` updates any file control blocks associated with the file. Exchanging the contents of two files requires essentially the same access permissions as opening both files for writing.

The files whose data is to be exchanged must both reside on the same volume. If they do not, `FSpExchangeFiles` returns the result code `diffVolErr`.

To exchange the contents of files with named forks other than the data and resource forks, or of files larger than 2 GB, use the `FSExchangeObjects` (page 486) , `PBExchangeObjectsSync` (page 636) , or `PBExchangeObjectsAsync` (page 635) function.

Special Considerations

The "compatibility code," by which `FSpExchangeFiles` attempted to perform the file exchange itself if it suspected that the underlying filesystem did not have Exchange capability, has been removed in Mac OS 9 and X.

Because other programs may have access paths open to one or both of the files exchanged, your application should have exclusive read/write access permission (`fsRdWrPerm`) to both files before calling `FSpExchangeFiles`. Exclusive read/write access to both files will ensure that `FSpExchangeFiles` doesn't affect another application because it prevents other applications from obtaining write access to one or both of the files exchanged.

`FSpExchangeFiles` does not respect the file-locked attribute; it will perform the exchange even if one or both of the files are locked. Obtaining exclusive read/write access to both files before calling `FSpExchangeFiles` ensures that the files are unlocked because locked files cannot be opened with write access.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

FSpGetFInfo

Obtains the Finder information for a file. (Deprecated in Mac OS X v10.4. Use `FSGetCatalogInfo` (page 494) instead.)

```
OSErr FSpGetFInfo (
   const FSSpec *spec,
   FInfo *fndrInfo
);
```

Parameters

spec

A pointer to an `FSSpec` structure specifying the file. See `FSSpec` (page 840) for a description of the `FSSpec` data type.

fndrInfo

On return, a pointer to information used by the Finder. The `FSpGetFInfo` function returns the Finder information from the volume catalog entry for the specified file. The function provides only the original Finder information—the information in the `FInfo` or `DInfo` structures, not the information in the `FXInfo` or `DXInfo` structures. For a description of the `FInfo` structure, see the *Finder Interface Reference*.

Return Value

A result code. If the specified object is a folder, this function returns `fnfErr`. For other possible return values, see "File Manager Result Codes" (page 943).

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
`Files.h`

FSpMakeFSRef

Creates an `FSRef` for a file or directory, given an `FSSpec`. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
OSErr FSpMakeFSRef (
   const FSSpec *source,
   FSRef *newRef
);
```

Parameters

source

A pointer to the `FSSpec` for the file or directory. This parameter must point to a valid `FSSpec` for an existing file or directory; if it does not, the call will return `fnfErr`. See `FSSpec` (page 840) for a description of the `FSSpec` data type.

newRef

On input, a pointer to an `FSRef` structure. On return, a pointer to the `FSRef` for the file or directory specified in the `FSSpec` pointed to in the `source` parameter.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
To obtain an `FSSpec` from an `FSRef`, use the `FSGetCatalogInfo` (page 494) function.

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
QTCarbonShell

Declared In
`Files.h`

FSpOpenDF

Opens the data fork of a file. (Deprecated in Mac OS X v10.4. Use `FSOpenFork` (page 514) instead.)

```
OSErr FSpOpenDF (
   const FSSpec *spec,
   SInt8 permission,
   FSIORefNum *refNum
);
```

Parameters

spec

A pointer to an `FSSpec` structure specifying the file whose data fork is to be opened. See `FSSpec` (page 840) for a description of the `FSSpec` data type.

permission

A constant indicating the type of access with which to open the file's data fork. In most cases, you can simply set the `permission` parameter to `fsCurPerm`. Some applications request `fsRdWrPerm`, to ensure that they can both read from and write to a file. For a description of the types of access that you can request, see "File Access Permission Constants" (page 908).

refNum

On return, a pointer to the file reference number for accessing the open data fork.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

Before calling this function, you should call the `Gestalt` function to check that the function is available. If `FSpOpenDF` is not available, you can use the function `HOpenDF` (page 554) instead.

Note that if you wish to access named forks other than the data and resource forks, or forks larger than 2GB, you will need to use the `FSOpenFork` (page 514) function, or one of its corresponding parameter block calls, `PBOpenForkSync` (page 740) and `PBOpenForkAsync` (page 739). If you try to open a fork larger than 2GB with the `FSpOpenDF` function, you will receive an error message.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

FSpOpenRF

Opens the resource fork of a file. (Deprecated in Mac OS X v10.4. Use `FSOpenFork` (page 514) instead.)

```
OSErr FSpOpenRF (
   const FSSpec *spec,
   SInt8 permission,
   FSIORefNum *refNum
);
```

Parameters

spec

A pointer to an `FSSpec` structure specifying the file whose resource fork is to be opened. See `FSSpec` (page 840) for a description of the `FSSpec` data type.

permission

A constant indicating the type of access with which to open the file's resource fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 908).

refNum

On return, a pointer to the file reference number for accessing the open resource fork.

Return Value

A result code. See "File Manager Result Codes" (page 943). On some file systems, `FSpOpenRF` will return the error `eofErr` if you try to open the resource fork of a file for which no resource fork exists with read-only access.

Discussion

Before calling this function, you should call the `Gestalt` function to check that the function is available. If `FSpOpenRF` is not available, you can use the function `HOpenRF` (page 554) instead.

Note that if you wish to access named forks other than the data and resource forks, or forks larger than 2GB, you will need to use the `FSOpenFork` (page 514) function, or one of its corresponding parameter block calls, `PBOpenForkSync` (page 740) or `PBOpenForkAsync` (page 739). If you try to open a fork larger than 2GB with the `FSpOpenRF` function, you will receive an error message.

Special Considerations

Generally, your application should use Resource Manager functions rather than File Manager functions to access a file's resource fork. The `FSpOpenRF` function does not read the resource map into memory and is generally useful only for applications (such as utilities that copy files) that need block-level access to a resource fork.

You should not use the resource fork of a file to hold non-resource data. Many parts of the system software assume that a resource fork always contains resource data.

Because there is no support for locking and unlocking file ranges on local disks in Mac OS X, regardless of whether File Sharing is enabled, you cannot open more than one path to a resource fork with read/write permission. If you try to open a more than one path to a file's resource fork with `fsRdWrShPerm` permission, only the first attempt will succeed. Subsequent attempts will return an invalid reference number and the `ResError` function will return the error `opWrErr`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

FSpRename

Renames a file or directory. (Deprecated in Mac OS X v10.4. Use `FSRenameUnicode` (page 539) instead.)

```
OSErr FSpRename (
   const FSSpec *spec,
   ConstStr255Param newName
);
```

Parameters

spec

A pointer to an `FSSpec` structure specifying the file or directory to rename. See `FSSpec` (page 840) for a description of the `FSSpec` data type.

newName

The new name of the file or directory.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
If a file ID reference for the specified file exists, it remains with the renamed file.

If you want to change the name of a new copy of an existing file, you should use the `FSpExchangeFiles` (page 528) function instead. To rename a file or directory using a long Unicode name, use the `FSRenameUnicode` (page 539) function or one of the corresponding parameter block calls, `PBRenameUnicodeSync` (page 748) and `PBRenameUnicodeAsync` (page 748).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

FSpRstFLock

Unlocks a file or directory. (Deprecated in Mac OS X v10.4. Use `FSSetCatalogInfo` (page 540) instead.)

```
OSErr FSpRstFLock (
   const FSSpec *spec
);
```

Parameters

spec

A pointer to an `FSSpec` structure specifying the file to unlock. See `FSSpec` (page 840) for a description of the `FSSpec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
If the `PBHGetVolParmsSync` (page 695) or `PBHGetVolParmsAsync` (page 694) function indicates that the volume supports folder locking (that is, the `bHasFolderLock` bit of the `vMAttrib` field is set), you can use `FSpRstFLock` to unlock a directory. Otherwise, you can only use this function to unlock a file.

You can lock a file or directory with the `FSpSetFLock` (page 535) , `HSetFLock` (page 558) , `PBHSetFLockSync` (page 724) , and `PBHSetFLockAsync` (page 723) 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
`Files.h`

FSpSetFInfo

Sets the Finder information about a file. (Deprecated in Mac OS X v10.4. Use `FSSetCatalogInfo` (page 540) instead.)

```
OSErr FSpSetFInfo (
   const FSSpec *spec,
   const FInfo *fndrInfo
);
```

Parameters

spec

A pointer to an `FSSpec` structure specifying the file for which to set the Finder information. See `FSSpec` (page 840) for a description of the `FSSpec` data type.

fndrInfo

A pointer to the new Finder information. For a description of the `FInfo` data type, see the *Finder Interface Reference*.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The `FSpSetFInfo` function changes the Finder information in the volume catalog entry for the specified file. `FSpSetFInfo` allows you to set only the original Finder information—the information in the `FInfo` or `DInfo` structures, not the information in the `FXInfo` or `DXInfo` 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
CarbonSketch
QTCarbonShell

Declared In
`Files.h`

FSpSetFLock

Locks a file or directory. (Deprecated in Mac OS X v10.4. Use `FSSetCatalogInfo` (page 540) instead.)

```
OSErr FSpSetFLock (
   const FSSpec *spec
);
```

Parameters

spec

A pointer to an `FSSpec` structure specifying the file or directory to lock. See `FSSpec` (page 840) for a description of the `FSSpec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
If the `PBHGetVolParmsSync` (page 695) or `PBHGetVolParmsAsync` (page 694) functions indicate that the volume supports folder locking (that is, the `bHasFolderLock` bit of the `vMAttrib` field is set), you can use `FSpSetFLock` to lock a directory. Otherwise, you can only use this function to lock a file.

After you lock a file, all new access paths to that file are read-only. This function has no effect on existing access paths.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

FSRead

Reads any number of bytes from an open file. (Deprecated in Mac OS X v10.4. Use `FSReadFork` (page 537) instead.)

```
OSErr FSRead (
    FSIORefNum refNum,
    SInt32 *count,
    void *buffPtr
);
```

Parameters

refNum

The file reference number of the open file from which to read.

count

On input, a pointer to the number of bytes to read; on output, a pointer to the number of bytes actually read.

buffPtr

A pointer to the data buffer into which the data will be read. This buffer is allocated by your application and must be at least as large as the `count` parameter.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
Because the read operation begins at the current mark, you might want to set the mark first by calling the `SetFPos` (page 787) function. If you try to read past the logical end-of-file, `FSRead` reads in all the data up to the end-of-file, moves the mark to the end-of-file, and returns `eofErr` as its function result. Otherwise, `FSRead` moves the file mark to the byte following the last byte read and returns `noErr`.

The low-level functions `PBReadSync` and `PBReadAsync` let you set the mark without having to call `SetFPos`. Furthermore, if you want to read data in newline mode, you must use `PBReadSync` or `PBReadAsync` instead of `FSRead`.

If you wish to read from named forks other than the data or resource forks, or from files larger than 2GB, you must use the `FSReadFork` (page 537) function, or one of its corresponding parameter block calls, `PBReadForkSync` (page 745) and `PBReadForkAsync` (page 744). If you attempt to use `FSRead` to read from a file larger than 2GB, you will receive an error message.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

FSReadFork

Reads data from an open fork.

```
OSErr FSReadFork (
    FSIORefNum forkRefNum,
    UInt16 positionMode,
    SInt64 positionOffset,
    ByteCount requestCount,
    void *buffer,
    ByteCount *actualCount
);
```

Parameters

forkRefNum
The reference number of the fork to read from. You should have previously opened this fork using the `FSOpenFork` (page 514) call, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 740) and `PBOpenForkAsync` (page 739).

positionMode

A constant specifying the base location within the fork for the start of the read. See "Position Mode Constants" (page 928) for a description of the constants which you can use to specify the base location.

The caller can also use this parameter to hint to the File Manager whether the data being read should or should not be cached. Caching reads appropriately can be important in ensuring that your program access files efficiently.

If you add the `forceReadMask` constant to the value you pass in this parameter, this tells the File Manager to force the data to be read directly from the disk. This is different from adding the `noCacheMask` constant since `forceReadMask` tells the File Manager to flush the appropriate part of the cache first, then ignore any data already in the cache. However, data that is read may be placed in the cache for future reads. The `forceReadMask` constant is also passed to the device driver, indicating that the driver should avoid reading from any device caches.

See "Cache Constants" (page 889) for further description of the constants that you can use to indicate your preference for caching the read.

positionOffset

The offset from the base location for the start of the read.

requestCount

The number of bytes to read.

buffer

A pointer to the buffer where the data will be returned.

actualCount

On return, a pointer to the number of bytes actually read. The value pointed to by the `actualCount` parameter should be equal to the value in the `requestCount` parameter unless there was an error during the read operation.

This parameter is optional; if you don't want this information returned, set `actualCount` to `NULL`.

Return Value

A result code. See "File Manager Result Codes" (page 943). If there are fewer than `requestCount` bytes from the specified position to the logical end-of-file, then all of those bytes are read, and `eofErr` is returned.

Discussion

`FSReadFork` reads data starting at the position specified by the `positionMode` and `positionOffset` parameters. The function reads up to `requestCount` bytes into the buffer pointed to by the `buffer` parameter and sets the fork's current position to the byte immediately after the last byte read (that is, the initial position plus `actualCount`).

To verify that data previously written has been correctly transferred to disk, read it back in using the `forceReadMask` constant in the `positionMode` parameter and compare it with the data you previously wrote.

When reading data from a fork, it is important to pay attention to that way that your program accesses the fork, because this can have a significant performance impact. For best results, you should use an I/O size of at least 4KB and block align your read requests. In Mac OS X, you should align your requests to 4KB boundaries.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

FSRefMakePath

Converts an `FSRef` structure into a POSIX-style pathname.

```
OSStatus FSRefMakePath (
   const FSRef *ref,
   UInt8 *path,
   UInt32 maxPathSize
);
```

Parameters

ref

A pointer to the `FSRef` structure to convert.

path

A pointer to a character buffer allocated by the caller. On output, the buffer contains a UTF-8 C string that specifies the absolute path to the object referred to by the *ref* parameter. The File Manager uses the *maxPathSize* parameter to make sure it does not overrun the buffer.

maxPathSize

The maximum number of bytes to copy into the buffer.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BSDLLCTest

Declared In

`Files.h`

FSRenameUnicode

Renames a file or folder.

```
OSErr FSRenameUnicode (
   const FSRef *ref,
   UniCharCount nameLength,
   const UniChar *name,
   TextEncoding textEncodingHint,
   FSRef *newRef
);
```

Parameters

ref

A pointer to an `FSRef` for the file or directory to rename. See `FSRef` (page 837) for a description of the `FSRef` data type.

nameLength

The length of the new name in Unicode characters.

name

A pointer to the new Unicode name of the file or directory.

textEncodingHint

The suggested text encoding to use when converting the Unicode name of the file or directory to some other encoding. If you pass the constant `kTextEncodingUnknown`, the File Manager will use a default value.

newRef

On return, a pointer to the new `FSRef` for the file or directory. This parameter is optional; if you do not wish the `FSRef` returned, pass `NULL`.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
Because renaming an object may change its `FSRef`, you should pass a non- `NULL` pointer in the `newRef` parameter and use the `FSRef` returned there to access the object after the renaming, if you wish to continue to refer to the object. The `FSRef` passed in the `ref` parameter may or may not be usable after the object is renamed. The `FSRef` returned in the `newRef` parameter may point to the same storage as the `FSRef` passed in `ref`.

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

Declared In
`Files.h`

FSSetCatalogInfo

Sets catalog information about a file or directory.

```
OSErr FSSetCatalogInfo (
   const FSRef *ref,
   FSCatalogInfoBitmap whichInfo,
   const FSCatalogInfo *catalogInfo
);
```

Parameters

ref

A pointer to an `FSRef` specifying the file or directory whose information is to be changed. See `FSRef` (page 837) for a description of the `FSRef` data type.

whichInfo

A bitmap specifying which catalog information fields to set. Only some of the catalog information fields may be set. These fields are given by the constant `kFSCatInfoSettableInfo`; no other bits may be set in the `whichInfo` parameter. See "Catalog Information Bitmap Constants" (page 891) for a description of the bits in this parameter.

To set the user ID (UID) and group ID (GID), specify the `kFSCatInfoSetOwnership` flag in this parameter. The File Manager attempts to set the user and group ID to the values specified in the `permissions` field of the catalog information structure. If `FSSetCatalogInfo` cannot set the user and group IDs, it returns an error.

catalogInfo

A pointer to the structure containing the new catalog information. Only some of the catalog information fields may be set. The fields which may be set are:

- createDate
- contentModDate
- attributeModDate
- accessDate
- backupDate
- permissions
- finderInfo
- extFinderInfo
- textEncodingHint

Return Value
A result code. See "File Manager Result Codes" (page 943).

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

Related Sample Code
BSDLLCTest

Declared In
Files.h

FSSetForkPosition

Sets the current position of an open fork.

```
OSErr FSSetForkPosition (
   FSIORefNum forkRefNum,
   UInt16 positionMode,
   SInt64 positionOffset
);
```

Parameters

forkRefNum

The reference number of a fork previously opened by the FSOpenFork (page 514), PBOpenForkSync (page 740), or PBOpenForkAsync (page 739) function.

positionMode

A constant specifying the base location within the fork for the new position. If this parameter is equal to fsAtMark, then the positionOffset parameter is ignored. See "Position Mode Constants" (page 928) for a description of the constants you can use to specify the base location.

positionOffset

The offset of the new position from the base location specified in the positionMode parameter.

Return Value
A result code. See "File Manager Result Codes" (page 943). This function returns the result code posErr if you attempt to set the current position of the fork to an offset before the start of the file.

Special Considerations

To determine if the `FSSetForkPosition` function is present, call the `Gestalt` function with the `gestaltFSAttr` selector. If the `FSSetForkPosition` function is present, but the volume does not directly support it, the File Manager will automatically call the `PBSetFPosSync` (page 764) function. However, if the volume does not directly support the `FSSetForkPosition` function, you can only set the file position for the data and resource forks, and you cannot grow these files beyond 2GB.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

FSSetForkSize

Changes the size of an open fork.

```
OSErr FSSetForkSize (
   FSIORefNum forkRefNum,
   UInt16 positionMode,
   SInt64 positionOffset
);
```

Parameters

forkRefNum

The reference number of the open fork. You can obtain this fork reference number with the `FSOpenFork` (page 514) function, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 740) and `PBOpenForkAsync` (page 739).

positionMode

A constant indicating the base location within the fork for the new size. See "Position Mode Constants" (page 928) for more information about the constants you can use to specify the base location.

positionOffset

The offset of the new size from the base location specified in the `positionMode` parameter.

Return Value

A result code. See "File Manager Result Codes" (page 943). If there is not enough space on the volume to extend the fork, then `dskFulErr` is returned and the fork's size is unchanged.

Discussion

The `FSSetForkSize` function sets the logical end-of-file to the position indicated by the `positionMode` and `positionOffset` parameters. The fork's new size may be less than, equal to, or greater than the fork's current size. If the fork's new size is greater than the fork's current size, then the additional bytes, between the old and new size, will have an undetermined value.

If the fork's current position is larger than the fork's new size, then the current position will be set to the new fork size the current position will be equal to the logical end-of-file.

Special Considerations

You do not need to check that the volume supports the `FSSetForkSize` function. If a volume does not support the `FSSetForkSize` function, but the `FSSetForkSize` function is present, the File Manager automatically calls the `PBSetEOFSync` (page 758) function and translates between the calls appropriately.

Note, however, that if the volume does not support the `FSSetForkSize` function, you can only access the data and resource forks, and you cannot grow the fork beyond 2GB. To check that the `FSSetForkSize` function is present, call the `Gestalt` function.

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

Declared In
`Files.h`

FSSetVolumeInfo

Sets information about a volume.

```
OSErr FSSetVolumeInfo (
   FSVolumeRefNum volume,
   FSVolumeInfoBitmap whichInfo,
   const FSVolumeInfo *info
);
```

Parameters

`volume`

The volume reference number of the volume whose information is to be changed. See `FSVolumeRefNum` (page 847) for a description of the `FSVolumeRefNum` data type.

`whichInfo`

A bitmap specifying which information to set. Only some of the volume information fields may be set. The settable fields are given by the constant `kFSVolInfoSettableInfo`; no other bits may be set in `whichInfo`. The fields which may be set are the backupDate, finderInfo, and `flags` fields. See "Volume Information Bitmap Constants" (page 938) for a description of the bits in this parameter.

`info`

A pointer to the new volume information. See `FSVolumeInfo` (page 842) for a description of the `FSVolumeInfo` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
To set information about the root directory of a volume, use the `FSSetCatalogInfo` (page 540) function, or one of the corresponding parameter block calls, `PBSetCatalogInfoSync` (page 753) and `PBSetCatalogInfoAsync` (page 751).

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

Declared In
`Files.h`

FSUnlockRange

Unlocks a range of bytes of the specified fork.

```
OSStatus FSUnlockRange (
    FSIORefNum forkRefNum,
    UInt16 positionMode,
    SInt64 positionOffset,
    UInt64 requestCount,
    UInt64 *rangeStart
);
```

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

Declared In
`Files.h`

FSUnmountVolumeAsync

Unmounts a volume asynchronously.

```
OSStatus FSUnmountVolumeAsync (
   FSVolumeRefNum vRefNum,
   OptionBits flags,
   FSVolumeOperation volumeOp,
   void *clientData,
   FSVolumeUnmountUPP callback,
   CFRunLoopRef runloop,
   CFStringRef runloopMode
);
```

Parameters

vRefNum
The volume reference number of the volume to unmount.

flags
Options for future use.

volumeOp
An `FSVolumeOperation` returned by the `FSCreateVolumeOperation` function.

clientData
A pointer to client data associated with the operation.

callback
The function to call when the unmount is complete.

runloop
The runloop to run on.

runloopMode
The mode for the runloop.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
This function starts the process of unmounting the volume specified by the *vRefNum* parameter. If a callback function is provided, that function will be called when the unmount operation is complete. Once this function returns `noErr` the status of the operation can be found using the `FSGetAsyncUnmountStatus` function.

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

Declared In
`Files.h`

FSUnmountVolumeSync

Unmounts a volume.

```
OSStatus FSUnmountVolumeSync (
   FSVolumeRefNum vRefNum,
   OptionBits flags,
   pid_t *dissenter
);
```

Parameters

vRefNum
The volume reference number of the volume to unmount.

flags
Options for future use.

dissenter
On return, a pointer to the pid of the process which denied the unmount if the unmount is denied.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
This function unmounts the volume specified by the *vRefNum* parameter. If the volume cannot be unmounted the pid of the process which denied the unmount will be returned in the *dissenter* parameter. This function returns after the unmount is complete.

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

Declared In
`Files.h`

FSVolumeMount

Mounts a volume using the specified mounting information.

```
OSStatus FSVolumeMount (
   BytePtr buffer,
   FSVolumeRefNum *mountedVolume
);
```

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

Declared In
`Files.h`

FSWrite

Writes any number of bytes to an open file. (Deprecated in Mac OS X v10.4. Use `FSWriteFork` (page 546) instead.)

```
OSErr FSWrite (
    FSIORefNum refNum,
    SInt32 *count,
    const void *buffPtr
);
```

Parameters

refNum

The file reference number of the open file to which to write.

count

On input, a pointer to the number of bytes to write to the file. Passing 0 in this parameter will return a `paramErr` error.

On output, a pointer to the number of bytes actually written.

buffPtr

A pointer to the data buffer containing the data to write.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The `FSWrite` function takes the specified number of bytes from the data buffer and attempts to write them to the file. Because the write operation begins at the current mark, you might want to set the mark first by calling the `SetFPos` (page 787) function.

If the write operation completes successfully, `FSWrite` moves the file mark to the byte following the last byte written and returns `noErr`. If you try to write past the logical end-of-file, `FSWrite` moves the logical end-of-file. If you try to write past the physical end-of-file, `FSWrite` adds one or more clumps to the file and moves the physical end-of-file accordingly.

The low-level functions `PBWriteSync` and `PBWriteAsync` let you set the mark without having to call `SetFPos`.

If you wish to write to named forks other than the data or resource forks, or grow files larger than 2GB, you must use the `FSWriteFork` (page 546) function, or one of its corresponding parameter block calls, `PBWriteForkSync` (page 777) and `PBWriteForkAsync` (page 776). If you attempt to use `FSWrite` to write to a file larger than 2GB, you will receive an error message.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

FSWriteFork

Writes data to an open fork.

```
OSErr FSWriteFork (
    FSIORefNum forkRefNum,
    UInt16 positionMode,
    SInt64 positionOffset,
    ByteCount requestCount,
    const void *buffer,
    ByteCount *actualCount
);
```

Parameters

forkRefNum

The reference number of the fork to which to write. You should have previously opened the fork using the `FSOpenFork` (page 514) function, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 740) and `PBOpenForkAsync` (page 739).

positionMode

A constant specifying the base location within the fork for the start of the write. See "Position Mode Constants" (page 928) for a description of the constants which you can use to specify the base location.

The caller can also use this parameter to hint to the File Manager whether the data being written should or should not be cached. See "Cache Constants" (page 889) for further description of the constants that you can use to indicate your preference for caching.

positionOffset

The offset from the base location for the start of the write.

requestCount

The number of bytes to write.

buffer

A pointer to a buffer containing the data to write.

actualCount

On return, a pointer to the number of bytes actually written. The value pointed to by the `actualCount` parameter will be equal to the value in the `requestCount` parameter unless there was an error during the write operation.

This parameter is optional; if you don't want this information, set `actualCount` to `NULL`.

Return Value

A result code. See "File Manager Result Codes" (page 943). If there is not enough space on the volume to write `requestCount` bytes, then `dskFulErr` is returned.

Discussion

`FSWriteFork` writes data starting at the position specified by the `positionMode` and `positionOffset` parameters. The function attempts to write `requestCount` bytes from the buffer pointed at by the `buffer` parameter and sets the fork's current position to the byte immediately after the last byte written (that is, the initial position plus `actualCount`).

When writing data to a fork, it is important to pay attention to that way that your program accesses the fork, because this can have a significant performance impact. For best results, you should use an I/O size of at least 4KB and block align your write requests. In Mac OS X, you should align your requests to 4KB boundaries.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

GetEOF

Determines the current logical size of an open file. (Deprecated in Mac OS X v10.4. Use `FSGetForkSize` (page 499) instead.)

```
OSErr GetEOF (
   FSIORefNum refNum,
   SInt32 *logEOF
);
```

Parameters

refNum

The file reference number of an open file.

logEOF

On return, a pointer to the logical size (the logical end-of-file) of the given file.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

To determine the size of a named fork other than the data or resource forks, or of a fork larger than 2 GB, use the `FSGetForkSize` (page 499) function, or one of the corresponding parameter block functions, `PBGetForkSizeSync` (page 665) and `PBGetForkSizeAsync` (page 664).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

GetFPos

Returns the current position of the file mark. (Deprecated in Mac OS X v10.4. Use `FSGetForkPosition` (page 499) instead.)

```
OSErr GetFPos (
   FSIORefNum refNum,
   SInt32 *filePos
);
```

Parameters

refNum

The file reference number of an open file.

filePos

On return, a pointer to the current position of the mark. The position value is zero-based; that is, the value of `filePos` is 0 if the file mark is positioned at the beginning of the file.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion
Because the read and write operations performed by the functions `FSRead` (page 536) and `FSWrite` (page 546) begin at the current mark, you should call `GetFPos`, or one of the parameter block functions, `PBGetFPosSync` (page 666) and `PBGetFPosAsync` (page 666) , to determine the current position of the file mark before reading from or writing to the file.

To determine the current position of a named fork, or of a fork larger than 2GB, use the `FSGetForkPosition` (page 499) function, or one of the corresponding parameter block calls, `PBGetForkPositionSync` (page 663) and `PBGetForkPositionAsync` (page 663).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

GetVRefNum

Gets a volume reference number from a file reference number. (Deprecated in Mac OS X v10.4. Use `FSGetCatalogInfo` (page 494) instead.)

```
OSErr GetVRefNum (
   FSIORefNum fileRefNum,
   FSVolumeRefNum *vRefNum
);
```

Parameters

`fileRefNum`
The file reference number of an open file.

`vRefNum`
On return, a pointer to the volume reference number of the volume containing the file specified in the `refNum` parameter.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
If you also want to determine the directory ID of the specified file's parent directory, call the `PBGetFCBInfoSync` (page 658) or `PBGetFCBInfoAsync` (page 656) 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
`Files.h`

HCreate

Creates a new file. (Deprecated in Mac OS X v10.4. Use `FSCreateFileUnicode` (page 481) instead.)

```
OSErr HCreate (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName,
   OSType creator,
   OSType fileType
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

The directory ID of the directory in which to create the new file.

fileName

The name of the new file. This can be a full or partial pathname.

You should not allow users to give files names that begin with a period (.). This ensures that files can be successfully opened by applications calling `HOpen` (page 553) instead of `HOpenDF` (page 554).

creator

The creator of the new file. For more information on a file's creator, see the Finder Interface documentation.

fileType

The file type of the new file. For more information on a file's type, see the Finder Interface documentation.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The `HCreate` function creates a new file (both data and resource forks) with the specified name, creator, and file type. The new file is unlocked and empty. The date and time of its creation and last modification are set to the current date and time.

Files created using `HCreate` are not automatically opened. If you want to write data to the new file, you must first open the file using a file access function.

The resource fork of the new file exists but is empty. You'll need to call one of the Resource Manager functions `HCreateResFile` or `FSpCreateResFile` to create a resource map in the file before you can open it (by calling one of the Resource Manager functions `HOpenResFile` or `FSpOpenResFile`).

To create a file with a Unicode filename, use the function `FSCreateFileUnicode` (page 481) , or one of the corresponding parameter block calls, `PBCreateFileUnicodeSync` (page 593) and `PBCreateFileUnicodeAsync` (page 591).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

HDelete

Deletes a file or directory. (Deprecated in Mac OS X v10.4. Use `FSDeleteObject` (page 484) instead.)

```
OSErr HDelete (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

The directory ID of the parent directory of the file or directory to delete.

fileName

The name of the file or directory to delete. This can be a full or partial pathname.

Return Value

A result code. See "File Manager Result Codes" (page 943). If you attempt to delete an open file or a non-empty directory, `HDelete` returns the result code `fBsyErr`. `HDelete` also returns the result code `fBsyErr` if the directory has an open working directory associated with it.

Discussion

If the specified target is a file, both the data and the resource fork of the file are deleted. In addition, if a file ID reference for the specified file exists, that reference is removed. A file must be closed before you can delete it. Similarly, you cannot delete a directory unless it's empty.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

HGetFInfo

Obtains the Finder information for a file. (Deprecated in Mac OS X v10.4. Use `FSGetCatalogInfo` (page 494) instead.)

```
OSErr HGetFInfo (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName,
   FInfo *fndrInfo
);
```

Parameters

vRefNum

The volume reference number, drive number, or 0 for the default volume.

dirID

The parent directory ID of the file.

fileName
The name of the file.

fndrInfo
On return, a pointer to the Finder information stored in the specified volume's catalog. The function returns only the original Finder information—that contained in an `FInfo` structure, not that in an `FXInfo` structure.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

HGetVol

Determines the current default volume and default directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr HGetVol (
   StringPtr volName,
   FSVolumeRefNum *vRefNum,
   SInt32 *dirID
);
```

Parameters
volName
On return, a pointer to the name of the default volume. If you do not want the name of the default volume returned, set this parameter to `NULL`.

vRefNum
On return, a pointer to the volume reference number of the default volume.

dirID
On return, a pointer to the directory ID of the default directory.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Version Notes
When CarbonLib is not present, the `HGetVol` function returns a working directory reference number in the `vRefNum` parameter if the previous call to `HSetVol` (page 559) (or one of the corresponding parameter block calls) passed in a working directory reference number.

Carbon Porting Notes

Carbon applications should use `HGetVol` and `HSetVol` to get and set the default directory. the functions `GetVol` and `SetVol`, as well as working directories, are no longer supported.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
Files.h

HOpen

Opens the data fork of a file. (Deprecated in Mac OS X v10.4. Use FSOpenFork (page 514) instead.)

```
OSErr HOpen (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName,
   SInt8 permission,
   FSIORefNum *refNum
);
```

Parameters

vRefNum
A volume reference number, drive number, or 0 for the default volume.

dirID
The directory ID of the file's parent directory.

fileName
The name of the file.

permission
A constant specifying the type of access with which to open the file's data fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 908).

refNum
On return, a pointer to the file reference number for accessing the open fork.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
If you use HOpen to try to open a file whose name begins with a period, you might mistakenly open a driver instead; subsequent attempts to write data might corrupt data on the target device. To avoid these problems, you should always use HOpenDF instead of HOpen.

Special Considerations

If you use HOpen to try to open a file whose name begins with a period, you might mistakenly open a driver instead; subsequent attempts to write data might corrupt data on the target device. To avoid these problems, you should always use HOpenDF (page 554) instead of HOpen.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Files.h

HOpenDF

Opens the data fork of a file. (Deprecated in Mac OS X v10.4. Use `FSOpenFork` (page 514) instead.)

```
OSErr HOpenDF (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName,
   SInt8 permission,
   FSIORefNum *refNum
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

The directory ID of the file's parent directory.

fileName

The name of the file.

permission

A constant specifying the type of access with which to open the file's data fork. For a description of the types of access which you can request, see "File Access Permission Constants" (page 908).

refNum

On return, a pointer to the file reference number for accessing the open data fork.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

Note that if you wish to access named forks other than the data and resource forks, or forks larger than 2GB, you will need to use the `FSOpenFork` (page 514) function, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 740) or `PBOpenForkAsync` (page 739). If you try to open a fork larger than 2GB with the `HOpenDF` function, you will receive an error message.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

HOpenRF

Opens the resource fork of a file. (Deprecated in Mac OS X v10.4. Use `FSOpenFork` (page 514) instead.)

```
OSErr HOpenRF (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName,
   SInt8 permission,
   FSIORefNum *refNum
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

The directory ID for the file's parent directory.

fileName

The name of the file.

permission

A constant specifying the type of access with which to open the file's resource fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 908).

refNum

On return, a pointer to the file reference number for accessing the open resource fork.

Return Value

A result code. See "File Manager Result Codes" (page 943). If you try to open the resource fork of a file for which no resource fork exists with read-only access, `HOpenRF` returns the error `eofErr`.

Discussion

Note that if you wish to access named forks other than the data and resource forks, or forks larger than 2GB, you will need to use the `FSOpenFork` (page 514) function, or one of its corresponding parameter block calls, `PBOpenForkSync` (page 740) or `PBOpenForkAsync` (page 739). If you try to open a fork larger than 2GB with the `HOpenRF` function, you will receive an error message.

Special Considerations

Generally, your application should use Resource Manager functions rather than File Manager functions to access a file's resource fork. The `HOpenRF` function does not read the resource map into memory and is generally useful only for applications (such as utilities that copy files) that need block-level access to a resource fork.

You should not use the resource fork of a file to hold non-resource data. Many parts of the system software assume that a resource fork always contains resource 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

`Files.h`

HRename

Renames a file, directory, or volume. (Deprecated in Mac OS X v10.4. Use `FSRenameUnicode` (page 539) instead.)

```
OSErr HRename (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param oldName,
   ConstStr255Param newName
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

A directory ID.

oldName

An existing filename, directory name, or volume name.

newName

The new filename, directory name, or volume name.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

Given the name of a file or directory in the `oldName` parameter, `HRename` changes it to the name in the `newName` parameter. Given a volume name in the `oldName` parameter or a volume reference number in the `vRefNum` parameter, `HRename` changes the name of the volume to the name in `newName`. Access paths currently in use aren't affected by this function.

If a file ID reference exists for a file you are renaming, the file ID remains with the renamed file.

To rename a file or directory using a long Unicode name, use the `FSRenameUnicode` (page 539) function or one of the corresponding parameter block calls, `PBRenameUnicodeSync` (page 748) and `PBRenameUnicodeAsync` (page 748).

Special Considerations

You cannot use `HRename` to change the directory in which a file resides. To move a file or directory, use the `FSpCatMove` (page 524), `PBCatMoveSync` (page 576), or `PBCatMoveAsync` (page 575) functions.

If you're renaming a volume, make sure that both names end with a colon.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

HRstFLock

Unlocks a file or directory. (Deprecated in Mac OS X v10.4. Use `FSSetCatalogInfo` (page 540) instead.)

```
OSErr HRstFLock (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

The parent directory ID of the file or directory to unlock.

fileName

The name of the file or directory.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

If the `PBHGetVolParmsSync` (page 695) or `PBHGetVolParmsAsync` (page 694) function indicates that the volume supports folder locking (that is, the `bHasFolderLock` bit of the `vMAttrib` field is set), you can use `HRstFLock` to unlock a directory. Otherwise, you can only use this function to unlock a file.

You can lock a file or directory with the `FSpSetFLock` (page 535) , `HSetFLock` (page 558) , `PBHSetFLockSync` (page 724) , and `PBHSetFLockAsync` (page 723) 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

`Files.h`

HSetFInfo

Sets the Finder information for a file. (Deprecated in Mac OS X v10.4. Use `FSSetCatalogInfo` (page 540) instead.)

```
OSErr HSetFInfo (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName,
   const FInfo *fndrInfo
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

The parent directory ID of the file.

fileName

The name of the file.

fndrInfo

A pointer to the new Finder information. The function changes the Finder information stored in the volume's catalog. `HSetFInfo` changes only the original Finder information—that contained in an `FInfo` structure, not that contained in an `FXInfo` structure. For a description of the `FInfo` data type, see the *Finder Interface Reference*.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

HSetFLock

Locks a file or directory. (Deprecated in Mac OS X v10.4. Use `FSSetCatalogInfo` (page 540) instead.)

```
OSErr HSetFLock (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

The parent directory ID of the file or directory to lock.

fileName

The name of the file or directory.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
If the `PBHGetVolParmsSync` (page 695) or `PBHGetVolParmsAsync` (page 694) function indicates that the volume supports folder locking (that is, the `bHasFolderLock` bit of the `vMAttrib` field is set), you can use `HSetFLock` to lock a directory. Otherwise, you can only use this function to lock a file.

After you lock a file, all new access paths to that file are read-only. This function has no effect on existing access paths.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

HSetVol

Sets the default volume and the default directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr HSetVol (
   ConstStr63Param volName,
   FSVolumeRefNum vRefNum,
   SInt32 dirID
);
```

Parameters

volName

The name of a mounted volume or the partial pathname of a directory. This parameter can be `NULL`.

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

A directory ID.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The `HSetVol` function lets you specify the default directory by volume reference number or by directory ID.

Both the default volume and the default directory are used in calls made with no volume name, a volume reference number of 0, and a directory ID of 0.

Special Considerations

Use of the `HSetVol` function is discouraged if your application may execute in system software versions prior to version 7.0. Because the specified directory might not itself be a working directory, `HSetVol` records the default volume and directory separately, using the volume reference number of the volume and the actual directory ID of the specified directory. Subsequent calls to `GetVol` (or `PBGetVolSync` or `PBGetVolAsync`) return only the volume reference number, which will cause that volume's root directory (rather than the default directory, as expected) to be accessed.

Carbon Porting Notes

Carbon applications should use `HGetVol` and `HSetVol` to get and set the default directory. the functions `GetVol` and `SetVol`, as well as working directories, are no longer supported.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

InvokeFNSubscriptionUPP

Calls your directory change callback function.

```
void InvokeFNSubscriptionUPP (
   FNMessage message,
   OptionBits flags,
   void *refcon,
   FNSubscriptionRef subscription,
   FNSubscriptionUPP userUPP
);
```

Discussion
The File Manager calls this function to invoke the directory change function which you have provided for use after an asynchronous call has been completed. You should not need to use this function yourself. For more information on directory change functions, see `FNSubscriptionProcPtr` (page 789).

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

Declared In
`Files.h`

InvokeFSVolumeEjectUPP

Calls your volume ejection callback function.

```
void InvokeFSVolumeEjectUPP (
   FSVolumeOperation volumeOp,
   void *clientData,
   OSStatus err,
   FSVolumeRefNum volumeRefNum,
   pid_t dissenter,
   FSVolumeEjectUPP userUPP
);
```

Discussion
The File Manager calls this function to invoke the volume ejection function which you have provided for use after an asynchronous call has been completed. You should not need to use this function yourself. For more information on change notification functions, see `FSVolumeEjectProcPtr` (page 792).

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

Declared In
`Files.h`

InvokeFSVolumeMountUPP

Calls your volume mount callback function.

```
void InvokeFSVolumeMountUPP (
   FSVolumeOperation volumeOp,
   void *clientData,
   OSStatus err,
   FSVolumeRefNum mountedVolumeRefNum,
   FSVolumeMountUPP userUPP
);
```

Discussion
The File Manager calls this function to invoke the volume mount function which you have provided for use after an asynchronous call has been completed. You should not need to use this function yourself. For more information on change notification functions, see `FSVolumeMountProcPtr` (page 792).

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

Declared In
`Files.h`

InvokeFSVolumeUnmountUPP

Calls your volume unmount callback function.

```
void InvokeFSVolumeUnmountUPP (
   FSVolumeOperation volumeOp,
   void *clientData,
   OSStatus err,
   FSVolumeRefNum volumeRefNum,
   pid_t dissenter,
   FSVolumeUnmountUPP userUPP
);
```

Discussion
The File Manager calls this function to invoke the volume unmount function which you have provided for use after an asynchronous call has been completed. You should not need to use this function yourself. For more information on change notification functions, see `FSVolumeUnmountProcPtr` (page 793).

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

Declared In
`Files.h`

InvokeIOCompletionUPP

Calls your I/O completion callback function.

```
void InvokeIOCompletionUPP (
   ParmBlkPtr paramBlock,
   IOCompletionUPP userUPP
);
```

Discussion
The File Manager calls this function to invoke the I/O completion function which you have provided for use after an asynchronous call has been completed. You should not need to use this function yourself. For more information on I/O completion functions, see `IOCompletionProcPtr` (page 794).

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

Declared In
`Files.h`

NewFNSubscriptionUPP

Creates a new universal procedure pointer (UPP) to your directory change callback function.

```
FNSubscriptionUPP NewFNSubscriptionUPP (
   FNSubscriptionProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to a directory change callback function. For more information, see `FNSubscriptionProcPtr` (page 789).

Return Value
A UPP to your directory change callback function.

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

Declared In
`Files.h`

NewFSVolumeEjectUPP

Creates a new universal procedure pointer (UPP) to your volume ejection callback function.

```
FSVolumeEjectUPP NewFSVolumeEjectUPP (
   FSVolumeEjectProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to a volume ejection callback function. For more information, see `FSVolumeEjectProcPtr` (page 792).

Return Value
A UPP to your volume ejection callback function.

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

Declared In
`Files.h`

NewFSVolumeMountUPP

Creates a new universal procedure pointer (UPP) to your volume mount callback function.

```
FSVolumeMountUPP NewFSVolumeMountUPP (
   FSVolumeMountProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to a volume mount callback function. For more information, see `FSVolumeEjectProcPtr` (page 792).

Return Value
A UPP to your volume mount callback function.

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

Declared In
`Files.h`

NewFSVolumeUnmountUPP

Creates a new universal procedure pointer (UPP) to your volume unmount callback function.

```
FSVolumeUnmountUPP NewFSVolumeUnmountUPP (
   FSVolumeUnmountProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to a volume unmount callback function. For more information, see `FSVolumeUnmountProcPtr` (page 793).

Return Value
A UPP to your volume unmount callback function.

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

Declared In
`Files.h`

NewIOCompletionUPP

Creates a new universal procedure pointer (UPP) to your I/O completion callback function.

```
IOCompletionUPP NewIOCompletionUPP (
   IOCompletionProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your I/O completion callback function. For more information, see `IOCompletionProcPtr` (page 794).

Return Value

A UPP to your I/O completion callback function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBAllocateAsync

Allocates additional space on a volume to an open file. (Deprecated in Mac OS X v10.4. Use `PBAllocateForkAsync` (page 565) instead.)

```
OSErr PBAllocateAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `IOParam` (page 862) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 866) for a description of the `ParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`ioRefNum`

On input, a file reference number for the file to which to allocate additional blocks.

`ioReqCount`

On input, the number of bytes to allocate.

`ioActCount`

On output, the number of bytes actually allocated, rounded up to the nearest multiple of the allocation block size.

The PBAllocateAsync function adds ioReqCount bytes to the specified file and sets the physical end-of-file to 1 byte beyond the last block allocated. If there isn't enough empty space on the volume to satisfy the allocation request, PBAllocateAsync allocates the rest of the space on the volume and returns dskFulErr as its function result.

If the total number of requested bytes is unavailable, PBAllocateAsync allocates whatever space, contiguous or not, is available. To force the allocation of the entire requested space as a contiguous piece, call PBAllocContigAsync (page 569) instead.

The File Manager automatically allocates file blocks if you move the logical end-of-file past the physical end-of-file, and it automatically deallocates unneeded blocks from a file if you move the logical end-of-file to a position more than one allocation block before the current physical end-of-file. Consequently, you do not in general need to be concerned with allocating or deallocating file blocks. However, you can improve file block contiguity if you use the PBAllocateAsync function to preallocate file blocks. This is most useful if you know in advance how big a file is likely to become.

The space allocated with this function is not permanently assigned to the file until the file's logical end-of-file is changed to include the allocated space. When a file (or volume) is closed, the space beyond the file's logical EOF is made available for other purposes, even if previously allocated to the file with a call to this function. You can change the end-of-file by setting it with the SetEOF (page 786) function, or by writing data to the file with the FSWrite (page 546) function.

This function is not supported by all file systems; for example, volumes mounted by the AppleShare file system do not support this function. To allocate space for a file on any volume, use the SetEOF (page 786) function, or one of the related parameter block calls, PBSetEOFSync (page 758) and PBSetEOFAsync (page 757).

To allocate space for a file beyond 2 GB, use the FSAllocateFork (page 470) function, or one of the corresponding parameter block functions, PBAllocateForkSync (page 567) and PBAllocateForkAsync (page 565).

Special Considerations

In Mac OS 7.5.5 through Mac OS 8.5, if there is not enough space left on the volume to allocate the requested number of bytes, the PBAllocateAsync function does not return the number of bytes actually allocated in the ioActCount field.

To determine the remaining space on a volume before calling PBAllocateAsync, use the functions PBXGetVolInfoSync or PBXGetVolInfoAsync.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

Files.h

PBAllocateForkAsync

Allocates space on a volume to an open fork.

```
void PBAllocateForkAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See FSForkIOParam (page 833) for a description of the FSForkIOParam data type.

Discussion

The relevant fields of the parameter block are:

ioCompletion

On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 794).

ioResult

On output, the result code of the function.

forkRefNum

On input, the reference number of the open fork. You can obtain a fork reference number with the FSOpenFork (page 514) function, or with one of the corresponding parameter block calls, PBOpenForkSync (page 740) and PBOpenForkAsync (page 739).

allocationFlags

On input, a constant indicating how the new space should be allocated. See "Allocation Flags" (page 887) for a description of the constants which you can use in this field.

positionMode

On input, a constant specifying the base location within the fork for the start of the allocation. See "Position Mode Constants" (page 928) for more information on the constants which you can use to specify the base location.

positionOffset

On input, the offset from the base location of the start of the allocation.

allocationAmount

On input, the number of bytes to allocate. On output, the number of bytes actually allocated to the file. The number of bytes allocated may be smaller than the requested amount if some of the space was already allocated. The value returned in this field does not reflect any additional bytes that may have been allocated because space is allocated in terms of fixed units such as allocation blocks, or the use of a clump size to reduce fragmentation.

The PBAllocateForkAsync function attempts to allocate the number of requested bytes of physical storage starting at the offset specified by the positionMode and positionOffset fields. For volume formats that support preallocated space, you can later write to this range of bytes (including extending the size of the fork) without requiring an implicit allocation.

Any extra space allocated but not used will be deallocated when the fork is closed, using FSCloseFork (page 475) , PBCloseForkSync (page 583) , or PBCloseForkAsync (page 582) ; or when flushed, using FSFlushFork (page 490) , PBFlushForkSync (page 639) , or PBFlushForkAsync (page 638).

Availability

Available in Mac OS X v10.0 and later.

Declared In

Files.h

PBAllocateForkSync

Allocates space on a volume to an open fork.

```
OSErr PBAllocateForkSync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`forkRefNum`

On input, the reference number of the open fork. You can obtain a fork reference number with the `FSOpenFork` (page 514) function, or with one of the corresponding parameter block functions, `PBOpenForkSync` (page 740) and `PBOpenForkAsync` (page 739).

`allocationFlags`

On input, a constant indicating how the new space should be allocated. See "Allocation Flags" (page 887) for a description of the constants you can use in this field.

`positionMode`

On input, a constant specifying the base location within the fork for the start of the allocation. See "Position Mode Constants" (page 928) for more information on the constants which you can use to specify the base location.

`positionOffset`

On input, the offset from the base location of the start of the allocation.

`allocationAmount`

On input, the number of bytes to allocate. On output, the number of bytes actually allocated to the file. The number of bytes allocated may be smaller than the requested amount if some of the space was already allocated. The value returned in this field does not reflect any additional bytes that may have been allocated because space is allocated in terms of fixed units such as allocation blocks, or the use of a clump size to reduce fragmentation.

The `PBAllocateForkSync` function attempts to allocate the number of requested bytes of physical storage starting at the offset specified by the `positionMode` and `positionOffset` fields. For volume formats that support preallocated space, you can later write to this range of bytes (including extending the size of the fork) without requiring an implicit allocation.

Any extra space allocated but not used will be deallocated when the fork is closed, using `FSCloseFork` (page 475) , `PBCloseForkSync` (page 583) , or `PBCloseForkAsync` (page 582) ; or when flushed, using `FSFlushFork` (page 490) , `PBFlushForkSync` (page 639) , or `PBFlushForkAsync` (page 638).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBAllocateSync

Allocates additional space on a volume to an open file. (Deprecated in Mac OS X v10.4. Use PBAllocateForkSync (page 567) instead.)

```
OSErr PBAllocateSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the IOParam (page 862) variant of the basic File Manager parameter block. See ParamBlockRec (page 866) for a description of the ParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

ioRefNum
: On input, a file reference number for the file to which to allocate additional blocks.

ioReqCount
: On input, the number of bytes to allocate.

ioActCount
: On output, the number of bytes actually allocated, rounded up to the nearest multiple of the allocation block size.

The PBAllocateSync function adds ioReqCount bytes to the specified file and sets the physical end-of-file to 1 byte beyond the last block allocated. If there isn't enough empty space on the volume to satisfy the allocation request, PBAllocateSync allocates the rest of the space on the volume and returns dskFulErr as its function result.

If the total number of requested bytes is unavailable, PBAllocateSync allocates whatever space, contiguous or not, is available. To force the allocation of the entire requested space as a contiguous piece, call PBAllocContigSync (page 570) instead.

The File Manager automatically allocates file blocks if you move the logical end-of-file past the physical end-of-file, and it automatically deallocates unneeded blocks from a file if you move the logical end-of-file to a position more than one allocation block before the current physical end-of-file. Consequently, you do not in general need to be concerned with allocating or deallocating file blocks. However, you can improve file block contiguity if you use the PBAllocateSync function to preallocate file blocks. This is most useful if you know in advance how big a file is likely to become.

The space allocated with this function is not permanently assigned to the file until the file's logical end-of-file is changed to include the allocated space. When a file (or volume) is closed, the space beyond the file's logical EOF is made available for other purposes, even if previously allocated to the file with a call to this function. You can change the end-of-file by setting it with the SetEOF (page 786) function, or by writing data to the file with the FSWrite (page 546) function.

This function is not supported by all file systems; for example, volumes mounted by the AppleShare file system do not support this function. To allocate space for a file on any volume, use the SetEOF (page 786) function, or one of the related parameter block calls, PBSetEOFSync (page 758) and PBSetEOFAsync (page 757).

To allocate space for a file beyond 2 GB, use the `FSAllocateFork` (page 470) function, or one of the corresponding parameter block functions, `PBAllocateForkSync` (page 567) and `PBAllocateForkAsync` (page 565).

Special Considerations

In Mac OS 7.5.5 through Mac OS 8.5, if there is not enough space left on the volume to allocate the requested number of bytes, the `PBAllocateSync` function does not return the number of bytes actually allocated in the `ioActCount` field.

To determine the remaining space on a volume before calling `PBAllocateSync`, use the functions `PBXGetVolInfoSync` or `PBXGetVolInfoAsync`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBAllocContigAsync

Allocates additional contiguous space on a volume to an open file. (Deprecated in Mac OS X v10.4. Use `PBAllocateForkAsync` (page 565) instead.)

```
OSErr PBAllocContigAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `IOParam` (page 862) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 866) for a description of the `ParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`ioRefNum`

On input, a file reference number for the open file.

`ioReqCount`

On input, the number of bytes to allocate.

`ioActCount`

On output, the number of bytes actually allocated, rounded up to the nearest multiple of the allocation block size.

The PBAllocContigAsync function is identical to the PBAllocateAsync (page 564) function except that if there isn't enough contiguous empty space on the volume to satisfy the allocation request, PBAllocContigAsync does nothing and returns dskFulErr as its function result. If you want to allocate whatever space is available, even when the entire request cannot be filled by the allocation of a contiguous piece, call PBAllocateAsync (page 564) instead.

The File Manager automatically allocates file blocks if you move the logical end-of-file past the physical end-of-file, and it automatically deallocates unneeded blocks from a file if you move the logical end-of-file to a position more than one allocation block before the current physical end-of-file. Consequently, you do not in general need to be concerned with allocating or deallocating file blocks. However, you can improve file block contiguity if you use the PBAllocContigAsync function to preallocate file blocks. This is most useful if you know in advance how big a file is likely to become.

The space allocated with this function is not permanently assigned to the file until the file's logical end-of-file is changed to include the allocated space. When a file (or volume) is closed, the space beyond the file's logical EOF is made available for other purposes, even if previously allocated to the file with a call to this function. You can change the end-of-file by setting it with the SetEOF (page 786) function, or by writing data to the file with the FSWrite (page 546) function.

This function is not supported by all file systems; for example, volumes mounted by the AppleShare file system do not support this function. To allocate space for a file on any volume, use the SetEOF (page 786) function, or one of the related parameter block calls, PBSetEOFSync (page 758) and PBSetEOFAsync (page 757).

To allocate space for a file beyond 2 GB, use the FSAllocateFork (page 470) function, or one of the corresponding parameter block functions, PBAllocateForkSync (page 567) and PBAllocateForkAsync (page 565).

Special Considerations

In Mac OS 7.5.5 through Mac OS 8.5, when there is not enough space to allocate the requested number of bytes, PBAllocContigAsync does not return 0 in the ioActCount field, so your application cannot rely upon this 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

Files.h

PBAllocContigSync

Allocates additional contiguous space on a volume to an open file. (Deprecated in Mac OS X v10.4. Use PBAllocateForkSync (page 567) instead.)

```
OSErr PBAllocContigSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `IOParam` (page 862) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 866) for a description of the `ParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioRefNum`

On input, a file reference number for the open file.

`ioReqCount`

On input, the number of bytes to allocate.

`ioActCount`

On output, the number of bytes actually allocated, rounded up to the nearest multiple of the allocation block size.

The `PBAllocContigSync` function is identical to the `PBAllocateSync` (page 568) function except that if there isn't enough contiguous empty space on the volume to satisfy the allocation request, `PBAllocContigSync` does nothing and returns `dskFulErr` as its function result. If you want to allocate whatever space is available, even when the entire request cannot be filled by the allocation of a contiguous piece, call `PBAllocateSync` (page 568) instead.

The File Manager automatically allocates file blocks if you move the logical end-of-file past the physical end-of-file, and it automatically deallocates unneeded blocks from a file if you move the logical end-of-file to a position more than one allocation block before the current physical end-of-file. Consequently, you do not in general need to be concerned with allocating or deallocating file blocks. However, you can improve file block contiguity if you use the `PBAllocContigSync` function to preallocate file blocks. This is most useful if you know in advance how big a file is likely to become.

The space allocated with this function is not permanently assigned to the file until the file's logical end-of-file is changed to include the allocated space. When a file (or volume) is closed, the space beyond the file's logical EOF is made available for other purposes, even if previously allocated to the file with a call to this function. You can change the end-of-file by setting it with the `SetEOF` (page 786) function, or by writing data to the file with the `FSWrite` (page 546) function.

This function is not supported by all file systems; for example, volumes mounted by the AppleShare file system do not support this function. To allocate space for a file on any volume, use the `SetEOF` (page 786) function, or one of the related parameter block calls, `PBSetEOFSync` (page 758) and `PBSetEOFAsync` (page 757).

To allocate space for a file beyond 2 GB, use the `FSAllocateFork` (page 470) function, or one of the corresponding parameter block functions, `PBAllocateForkSync` (page 567) and `PBAllocateForkAsync` (page 565).

Special Considerations

In Mac OS 7.5.5 through Mac OS 8.5, when there is not enough space to allocate the requested number of bytes, `PBAllocContigSync` does not return 0 in the `ioActCount` field, so your application cannot rely upon this 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

`Files.h`

PBCatalogSearchAsync

Searches for objects traversed by a catalog iterator that match a given set of criteria.

```
void PBCatalogSearchAsync (
   FSCatalogBulkParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a catalog information parameter block. See `FSCatalogBulkParam` (page 824) for a description of the `FSCatalogBulkParam` data type.

Discussion

The relevant fields of this parameter are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function. When the entire volume has been searched, `errFSNoMoreItems` is returned.

`iterator`

On input, the iterator to use. Objects traversed by this iterator are matched against the criteria specified by the `searchParams` field. You can obtain a catalog iterator with the function `FSOpenIterator` (page 515) , or with one of the related parameter block calls, `PBOpenIteratorSync` (page 742) and `PBOpenIteratorAsync` (page 741). Currently, this iterator must be created with the `kFSIterateSubtree` option and the container must be the root directory of a volume. See `FSIterator` (page 835) for more information on the `FSIterator` data type.

`searchParams`

On input, a pointer to an `FSSearchParams` (page 839) structure containing the search criteria. You can match against the object's name in Unicode and by the fields in an `FSCatalogInfo` (page 826) structure. You may use the same search bits as passed in the `ioSearchBits` field to the `PBCatSearchSync` (page 580) and `PBCatSearchAsync` (page 577) functions; they control the corresponding `FSCatalogInfo` fields. See "Catalog Search Masks" (page 900) for a description of the search bits. There are a few new search criteria supported by `PBCatalogSearchAsync` but not by `PBCatSearchSync` and `PBCatSearchAsync`. These new search criteria are indicated by the constants described in "Catalog Search Constants" (page 899).If the `searchTime` field of this structure is non-zero,

it is interpreted as a Time Manager duration; the search may terminate after this duration even if `maximumItems` objects have not been returned and the entire catalog has not been scanned. If `searchTime` is zero, there is no time limit for the search. If you are searching by any criteria other than name, you must set the `searchInfo1` and `searchInfo2` fields of the structure in this field to point to `FSCatalogInfo` structures containing the values to match against.

`maximumItems`
On input, the maximum number of items to return for this call.

`actualItems`
On output, the actual number of items returned for this call.

`containerChanged`
On output, a Boolean value indicating whether the container's contents have changed. If `true`, the container's contents changed since the previous `PBCatalogSearchAsync` call. Objects may still be returned even though the container changed. Note that if the container has changed, then the total set of items returned may be incorrect; some items may be returned multiple times, and some items may not be returned at all.

`whichInfo`
On input, a bitmap specifying the catalog information fields to return for each item. If you don't wish any catalog information returned, pass the constant `kFSCatInfoNone` in this field. See "Catalog Information Bitmap Constants" (page 891) for a description of the bits in this field.

`catalogInfo`
On output, a pointer to an array of `FSCatalogInfo` (page 826) structures; one for each found item. On input, the `catalogInfo` field should point to an array of `maximumItems` catalog information structures. This field is optional; if you do not wish any catalog information returned, pass `NULL` here.

`refs`
On output, a pointer to an array of `FSRef` (page 837) structures; one for each returned item. On input, if you want an `FSRef` for each item found, pass a pointer to an array of `maximumItems FSRef` structures. Otherwise, pass `NULL`.

`names`
On output, a pointer to an array of filenames; one for each returned item. On input, if you want the Unicode filename for each item found, pass a pointer to an array of `maximumItems HFSUniStr255` (page 855) structures. Otherwise, pass `NULL`.

A single search may span more than one call to `PBCatalogSearchAsync`. The call may complete with no error before scanning the entire volume. This typically happens because the time limit (`searchTime`) has been reached or `maximumItems` items have been returned. If the search is not completed, you can continue the search by making another call to `PBCatalogSearchAsync` and passing the updated iterator returned by the previous call in the `iterator` field.

Before calling this function, you should determine that it is present, by calling the `Gestalt` function.

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

Declared In
`Files.h`

PBCatalogSearchSync

Searches for objects traversed by a catalog iterator that match a given set of criteria.

```
OSErr PBCatalogSearchSync (
   FSCatalogBulkParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a catalog information parameter block. See `FSCatalogBulkParam` (page 824) for a description of the `FSCatalogBulkParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943). When the entire volume has been searched, `errFSNoMoreItems` is returned.

Discussion

The relevant fields of this parameter are:

`iterator`

On input, the iterator to use. Objects traversed by this iterator are matched against the criteria specified by the `searchParams` field. You can obtain a catalog iterator with the function `FSOpenIterator` (page 515) , or with one of the related parameter block calls, `PBOpenIteratorSync` (page 742) and `PBOpenIteratorAsync` (page 741). Currently, this iterator must be created with the `kFSIterateSubtree` option and the container must be the root directory of a volume. See `FSIterator` (page 835) for more information on the `FSIterator` data type.

`searchParams`

On input, a pointer to an `FSSearchParams` (page 839) structure containing the search criteria. You can match against the object's name in Unicode and by the fields in an `FSCatalogInfo` (page 826) structure. You may use the same search bits as passed in the `ioSearchBits` field to the `PBCatSearchSync` (page 580) and `PBCatSearchAsync` (page 577) functions; they control the corresponding `FSCatalogInfo` fields. See "Catalog Search Masks" (page 900) for a description of the search bits. There are a few new search criteria supported by `PBCatalogSearchSync` but not by `PBCatSearchSync` and `PBCatSearchAsync`. These new search criteria are indicated by the constants described in "Catalog Search Constants" (page 899).If the `searchTime` field of this structure is non-zero, it is interpreted as a Time Manager duration; the search may terminate after this duration even if `maximumItems` objects have not been returned and the entire catalog has not been scanned. If `searchTime` is zero, there is no time limit for the search. If you are searching by any criteria other than name, you must set the `searchInfo1` and `searchInfo2` fields of the structure in this field to point to `FSCatalogInfo` structures containing the values to match against.

`maximumItems`

On input, the maximum number of items to return for this call.

`actualItems`

On output, the actual number of items returned for this call.

`containerChanged`

On output, a Boolean value indicating whether the container's contents have changed. If `true`, the container's contents changed since the previous `PBCatalogSearchSync` call. Objects may still be returned even though the container changed. Note that if the container has changed, then the total set of items returned may be incorrect; some items may be returned multiple times, and some items may not be returned at all.

`whichInfo`

On input, a bitmap specifying the catalog information fields to return for each item. If you don't wish any catalog information returned, pass the constant `kFSCatInfoNone` in this field. See "Catalog Information Bitmap Constants" (page 891) for a description of the bits in this field.

catalogInfo
: On output, a pointer to an array of FSCatalogInfo (page 826) structures; one for each found item. On input, the catalogInfo field should point to an array of maximumItems catalog information structures. This field is optional; if you do not wish any catalog information returned, pass NULL here.

refs
: On output, a pointer to an array of FSRef (page 837) structures; one for each returned item. On input, if you want an FSRef for each item found, pass a pointer to an array of maximumItems FSRef structures. Otherwise, pass NULL.

names
: On output, a pointer to an array of filenames; one for each returned item. On input, if you want the Unicode filename for each item found, pass a pointer to an array of maximumItems HFSUniStr255 (page 855) structures. Otherwise, pass NULL.

A single search may span more than one call to PBCatalogSearchSync. The call may complete with no error before scanning the entire volume. This typically happens because the time limit (searchTime) has been reached or maximumItems items have been returned. If the search is not completed, you can continue the search by making another call to PBCatalogSearchSync and passing the updated iterator returned by the previous call in the iterator field.

Before calling this function, you should determine that it is present, by calling the Gestalt function.

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

Declared In
Files.h

PBCatMoveAsync

Moves files or directories from one directory to another on the same volume. (Deprecated in Mac OS X v10.4. Use PBMoveObjectAsync (page 737) instead.)

```
OSErr PBCatMoveAsync (
   CMovePBPtr paramBlock
);
```

Parameters

paramBlock
: A pointer to a catalog move parameter block. See CMovePBRec (page 802) for a description of the CMovePBRec data type.

Return Value
A result code. See "File Manager Result Codes" (page 943). This function returns permErr if called on a locked file.

Discussion
The relevant fields of the parameter block are:

ioCompletion
: On input, a pointer to a completion function. For more information on completion routines, see IOCompletionProcPtr (page 794).

ioResult
: On output, the result code of the function.

`ioNamePtr`
: On input, a pointer to the name of the file or directory to move.

`ioVRefNum`
: On input, a volume reference number, drive number, or 0 for the default volume.

`ioNewName`
: On input, a pointer to the name of the destination directory. Pass `NULL` in this field if you wish to specify the destination directory by its directory ID.

`ioNewDirID`
: On input, if the `ioNewName` field is `NULL`, the directory ID of the destination directory. If `ioNewName` is not `NULL`, this is the parent directory ID of the directory into which the file or directory is to be moved. It is usually simplest to specify the destination directory by passing its directory ID in the `ioNewDirID` field and by setting `ioNewName` to `NULL`.

`ioDirID`
: On input, the parent directory ID of the file or directory to move.

`PBCatMoveAsync` is strictly a file catalog operation; it does not actually change the location of the file or directory on the disk. If a file ID reference exists for the file, the file ID reference remains with the moved file.

The `PBCatMoveAsync` function cannot move a file or directory to another volume (that is, the value in the `ioVRefNum` field is used in specifying both the source and the destination). Also, you cannot use it to rename files or directories; to rename a file or directory, use `FSpRename` (page 533) , `PBHRenameSync` (page 716) , or `PBHRenameAsync` (page 715) .

If you need to move files or directories with named forks other than the data and resource forks, with long Unicode names, or files larger than 2GB, you should use the `FSMoveObject` (page 510) function, or one of the corresponding parameter block calls, `PBMoveObjectSync` (page 738) and `PBMoveObjectAsync` (page 737).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBCatMoveSync

Moves files or directories from one directory to another on the same volume. (Deprecated in Mac OS X v10.4. Use `PBMoveObjectSync` (page 738) instead.)

```
OSErr PBCatMoveSync (
   CMovePBPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a catalog move parameter block. See `CMovePBRec` (page 802) for a description of the `CMovePBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943). This function returns `permErr` if called on a locked file.

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the name of the file or directory to move.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioNewName`
On input, a pointer to the name of the destination directory. Pass `NULL` in this field if you wish to specify the destination directory by its directory ID.

`ioNewDirID`
On input, if the `ioNewName` field is `NULL`, the directory ID of the destination directory. If `ioNewName` is not `NULL`, this is the parent directory ID of the destination directory. It is usually simplest to specify the destination directory by passing its directory ID in the `ioNewDirID` field and by setting `ioNewName` to `NULL`.

`ioDirID`
On input, the parent directory ID of the file or directory to move.

`PBCatMoveSync` is strictly a file catalog operation; it does not actually change the location of the file or directory on the disk. If a file ID reference exists for the file, the file ID reference remains with the moved file.

The `PBCatMoveSync` function cannot move a file or directory to another volume (that is, the value in the `ioVRefNum` field is used in specifying both the source and the destination). Also, you cannot use it to rename files or directories; to rename a file or directory, use `FSpRename` (page 533) , `PBHRenameSync` (page 716) , or `PBHRenameAsync` (page 715).

If you need to move files or directories with named forks other than the data and resource forks, with long Unicode names, or files larger than 2GB, you should use the `FSMoveObject` (page 510) function, or one of the corresponding parameter block calls, `PBMoveObjectSync` (page 738) and `PBMoveObjectAsync` (page 737).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBCatSearchAsync

Searches a volume's catalog file using a set of search criteria that you specify. (Deprecated in Mac OS X v10.4. Use `PBCatalogSearchAsync` (page 572) instead.)

```
OSErr PBCatSearchAsync (
   CSParamPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a `CSParam` (page 807) variant of an HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function. When `PBCatSearchAsync` has searched the entire volume, it returns `eofErr`. If it exits because it either spends the maximum time allowed in the `ioSearchTime` field or finds the maximum number of matches allowed in the `ioReqMatchCount` field, it returns `noErr`.

`ioNamePtr`

On input, a pointer to the name of the volume to search.

`ioVRefNum`

On input, a volume reference number or drive number for the volume to search; or 0 for the default volume.

`ioMatchPtr`

On input, a pointer to an array of `FSSpec` (page 840) structure to hold the matches found by this function. On return, the `FSSpec` structures in this array identify the files and directories that match the criteria.

`ioReqMatchCount`

On input, the maximum number of matches to return.

`ioActMatchCount`

On output, the actual number of matches returned.

`ioSearchBits`

On input, a bitmap specifying the fields in the criteria structures to match against. See "Catalog Search Masks" (page 900) for a description of the bits in this field.

`ioSearchInfo1`

On input, a pointer to a `CInfoPBRec` (page 802) union containing search information. For values that match by mask and value (Finder information, for example), set the bits in the structure passed in `ioSearchInfo2`, and set the matching value in this structure. For values that match against a range (such as dates), set the lower bounds for the range in this structure.

`ioSearchInfo2`

On input, a pointer to a `CInfoPBRec` (page 802) union containing search information. For values that match by mask and value (Finder information, for example), set the bits in this structure, and set the matching value in the structure passed in the `ioSearchInfo1` field. For values that match against a range (such as dates), set the upper bounds for the range in this structure.

`ioSearchTime`
: On input, the maximum allowed search time. If you pass 0 in this field, no time limit is set.

`ioCatPosition`
: The current catalog position, specified as a `CatPositionRec` (page 801) structure. You can use this field, along with the `ioSearchTime` field, to search a volume in segments. To search a volume in segments, set a time limit for the search in the `ioSearchTime` field and set the `initialize` field of the `CatPositionRec` structure to the location for the start of the search (0 if you wish to start searching at the beginning of the volume). On return, the catalog position will be updated. You can then pass this updated `CatPositionRec` structure to the next call to `PBCatSearchSync` to continue searching at the place where you left off.

`ioOptBuffer`
: On input, a pointer to an optional read buffer.

`ioOptBufSize`
: On input, the length of the optional read buffer.

If the catalog file changes between two timed calls to `PBCatSearchAsync` (that is, when you are using `ioSearchTime` and `ioCatPosition` to search a volume in segments and the catalog file changes between searches), `PBCatSearchAsync` returns a result code of `catChangedErr` and no matches. Depending on what has changed on the volume, `ioCatPosition` might be invalid, most likely by a few entries in one direction or another. You can continue the search, but you risk either skipping some entries or reading some twice.

Special Considerations

Not all volumes support the `PBCatSearchAsync` function. Before you call `PBCatSearchAsync` to search a particular volume, you should call the `PBHGetVolParmsAsync` (page 694) function to determine whether that volume supports `PBCatSearchAsync`. If the `bHasCatSearch` bit is set in the `vMAttrib` field, then the volume supports `PBCatSearchAsync`.

Even though AFP volumes support `PBCatSearchSync`, they do not support all of its features that are available on local volumes. These restrictions apply to AFP volumes:

- AFP volumes do not use the `ioSearchTime` field. Current versions of the AppleShare server software search for 1 second or until 4 matches are found. The AppleShare workstation software keeps requesting the appropriate number of matches until the server returns either the number specified in the `ioReqMatchCount` field or an error.
- AFP volumes do not support both logical and physical fork lengths. If you request a search using the length of a fork, the actual minimum length used is the smallest of the values in the logical and physical fields of the `ioSearchInfo1` structure and the actual maximum length used is the largest of the values in the logical and physical fields of the `ioSearchInfo2` structure.
- The `fsSBNegate` bit of the `ioSearchBits` field is ignored during searches of remote volumes that support AFP version 2.1.
- If the AFP server returns `afpCatalogChanged`, the catalog position structure returned to your application (in the `ioCatPosition` field) is the same one you passed to `PBCatSearchAsync`. You should clear the `initialize` field of that structure to restart the search from the beginning.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Files.h`

PBCatSearchSync

Searches a volume's catalog file using a set of search criteria that you specify. (Deprecated in Mac OS X v10.4. Use `PBCatalogSearchSync` (page 573) instead.)

```
OSErr PBCatSearchSync (
   CSParamPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a `CSParam` (page 807) variant of an HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943). When `PBCatSearchSync` has searched the entire volume, it returns `eofErr`. If it exits because it either spends the maximum time allowed in the `ioSearchTime` field or finds the maximum number of matches allowed in the `ioReqMatchCount` field, it returns `noErr`.

Discussion

The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to the name of the volume to search.

`ioVRefNum`

On input, a volume reference number or drive number for the volume to search; or 0 for the default volume.

`ioMatchPtr`

On input, a pointer to an array of `FSSpec` (page 840) structure to hold the matches found by this function. On return, the `FSSpec` structures in this array identify the files and directories that match the criteria.

`ioReqMatchCount`

On input, the maximum number of matches to return.

`ioActMatchCount`

On output, the actual number of matches returned.

`ioSearchBits`

On input, a bitmap specifying the fields in the criteria structures to match against. See "Catalog Search Masks" (page 900) for a description of the bits in this field.

`ioSearchInfo1`

On input, a pointer to a `CInfoPBRec` (page 802) union containing search information. For values that match by mask and value (Finder information, for example), set the bits in the structure passed in `ioSearchInfo2`, and set the matching value in this structure. For values that match against a range (such as dates), set the lower bounds for the range in this structure.

`ioSearchInfo2`

On input, a pointer to a `CInfoPBRec` (page 802) union containing search information. For values that match by mask and value (Finder information, for example), set the bits in this structure, and set the

matching value in the structure passed in the `ioSearchInfo1` field. For values that match against a range (such as dates), set the upper bounds for the range in this structure.

`ioSearchTime`
On input, the maximum allowed search time. If you pass 0 in this field, no time limit is set.

`ioCatPosition`
The current catalog position, specified as a `CatPositionRec` (page 801) structure. You can use this field, along with the `ioSearchTime` field, to search a volume in segments. To search a volume in segments, set a time limit for the search in the `ioSearchTime` field and set the `initialize` field of the `CatPositionRec` structure to the location for the start of the search (0 if you wish to start searching at the beginning of the volume). On return, the catalog position will be updated. You can then pass this updated `CatPositionRec` structure to the next call to `PBCatSearchSync` to continue searching at the place where you left off.

`ioOptBuffer`
On input, a pointer to an optional read buffer.

`ioOptBufSize`
On input, the length of the optional read buffer.

If the catalog file changes between two timed calls to `PBCatSearchSync` (that is, when you are using `ioSearchTime` and `ioCatPosition` to search a volume in segments and the catalog file changes between searches), `PBCatSearchSync` returns a result code of `catChangedErr` and no matches. Depending on what has changed on the volume, `ioCatPosition` might be invalid, most likely by a few entries in one direction or another. You can continue the search, but you risk either skipping some entries or reading some twice.

Special Considerations

Not all volumes support the `PBCatSearchSync` function. Before you call `PBCatSearchSync` to search a particular volume, you should call the `PBHGetVolParmsSync` (page 695) function to determine whether that volume supports `PBCatSearchSync`.If the `bHasCatSearch` bit is set in the `vMAttrib` field, then the volume supports `PBCatSearchSync`.

Even though AFP volumes support `PBCatSearchSync`, they do not support all of its features that are available on local volumes. These restrictions apply to AFP volumes:

- AFP volumes do not use the `ioSearchTime` field. Current versions of the AppleShare server software search for 1 second or until 4 matches are found. The AppleShare workstation software keeps requesting the appropriate number of matches until the server returns either the number specified in the `ioReqMatchCount` field or an error.
- AFP volumes do not support both logical and physical fork lengths. If you request a search using the length of a fork, the actual minimum length used is the smallest of the values in the logical and physical fields of the `ioSearchInfo1` structure and the actual maximum length used is the largest of the values in the logical and physical fields of the `ioSearchInfo2` structure.
- The `fsSBNegate` bit of the `ioSearchBits` field is ignored during searches of remote volumes that support AFP version 2.1.
- If the AFP server returns `afpCatalogChanged`, the catalog position structure returned to your application (in the `ioCatPosition` field) is the same one you passed to `PBCatSearchSync`. You should clear the `initialize` field of that structure to restart the search from the beginning.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Files.h`

PBCloseAsync

Closes an open file. (Deprecated in Mac OS X v10.5. Use `PBCloseForkAsync` (page 582) instead.)

```
OSErr PBCloseAsync (
    ParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to a basic File Manager parameter block.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine.

`ioResult`
On output, the result code of the function.

`ioRefNum`
On input, a file reference number to the file to close.

The `PBCloseAsync` function writes the contents of the access path buffer specified by the `ioRefNum` field to the volume and removes the access path.

Special Considerations

Some information stored on the volume won't be updated until `PBFlushVolAsync` is called.

Do not call `PBCloseAsync` with a file reference number of a file that has already been closed. Attempting to close the same file twice may result in loss of data on a volume.

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
`Files.h`

PBCloseForkAsync

Closes an open fork.

```
void PBCloseForkAsync (
    FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam`.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`forkRefNum`

On input, the reference number of the fork to close. After the call to this function, the reference number in this parameter is invalid.

The `PBCloseForkAsync` function causes all data written to the fork to be written to disk, in the same manner as the `PBFlushForkAsync` (page 638) function, before it closes the fork.

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

Declared In
`Files.h`

PBCloseForkSync

Closes an open fork.

```
OSErr PBCloseForkSync (
    FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam`.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant field of the parameter block is:

`forkRefNum`

On input, the reference number of the fork to close. After the call to this function, the reference number in this parameter is invalid.

The `PBCloseForkSync` function causes all data written to the fork to be written to disk, in the same manner as the `PBFlushForkSync` (page 639) function, before it closes the fork.

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

Declared In
`Files.h`

PBCloseIteratorAsync

Closes a catalog iterator.

```
void PBCloseIteratorAsync (
   FSCatalogBulkParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a catalog information parameter block. See `FSCatalogBulkParam` (page 824) for a description of the `FSCatalogBulkParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`iterator`

On input, the catalog iterator to close. `PBCloseIteratorAsync` releases memory and other system resources used by the iterator, making the iterator invalid. See `FSIterator` (page 835) for a description of the `FSIterator` data type.

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

Declared In
`Files.h`

PBCloseIteratorSync

Closes a catalog iterator.

```
OSErr PBCloseIteratorSync (
   FSCatalogBulkParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a catalog information parameter block. See `FSCatalogBulkParam` (page 824) for a description of the `FSCatalogBulkParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant field of the parameter block is:

`iterator`
On input, the catalog iterator to close. `PBCloseIteratorSync` releases memory and other system resources used by the iterator, making the iterator invalid. See `FSIterator` (page 835) for a description of the `FSIterator` data type.

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

Declared In
`Files.h`

PBCloseSync

Closes an open file. (Deprecated in Mac OS X v10.5. Use `PBCloseForkSync` (page 583) instead.)

```
OSErr PBCloseSync (
    ParmBlkPtr paramBlock
);
```

Parameters
paramBlock
A pointer to a basic File Manager parameter block.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant field of the parameter block is:

`ioRefNum`
On input, a file reference number to the file to close.

The `PBCloseSync` function writes the contents of the access path buffer specified by the `ioRefNum` field to the volume and removes the access path.

Special Considerations

Some information stored on the volume won't be updated until `PBFlushVolSync` is called.

Do not call `PBCloseSync` with a file reference number of a file that has already been closed. Attempting to close the same file twice may result in loss of data on a volume.

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
`Files.h`

PBCompareFSRefsAsync

Determines whether two FSRef structures refer to the same file or directory.

```
void PBCompareFSRefsAsync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See FSRefParam (page 837) for a description of the FSRefParam data type.

Discussion

The relevant fields of the parameter block are:

ioCompletion

On input, a pointer to a completion routine. For more information about completion routines, see IOCompletionProcPtr (page 794).

ioResult

On output, the result code of the function. If the two FSRef structures refer to the same file or directory, then noErr is returned. If they refer to objects on different volumes, then diffVolErr is returned. If they refer to different files or directories on the same volume, then errFSRefsDifferent is returned. This call may return other errors, including nsvErr, fnfErr, dirNFErr, and volOffLinErr. See "File Manager Result Codes".

ref

On input, a pointer to the first FSRef to compare. See FSRef (page 837) for a description of the FSRef data type.

parentRef

On input, a pointer to the second FSRef to compare.

You must use FSCompareFSRefs (page 476) , or one of the corresponding parameter block functions, PBCompareFSRefsSync (page 586) and PBCompareFSRefsAsync, to compare FSRef structures. It is not possible to compare the FSRef structures directly since some bytes may be uninitialized, case-insensitive text, or contain hint information.

Some volume formats may be able to tell that two FSRef structures would refer to two different files or directories, without having to actually find those objects. In this case, the volume format may return errFSRefsDifferent even if one or both objects no longer exist. Similarly, if the FSRef structures are for objects on different volumes, the File Manager will return diffVolErr even if one or both volumes are no longer mounted.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Files.h

PBCompareFSRefsSync

Determines whether two FSRef structures refer to the same file or directory.

```
OSErr PBCompareFSRefsSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 837) for a description of the `FSRefParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943). If the two `FSRef` structures refer to the same file or directory, then `noErr` is returned. If they refer to objects on different volumes, then `diffVolErr` is returned. If they refer to different files or directories on the same volume, then `errFSRefsDifferent` is returned. This function may return other errors, including `nsvErr`, `fnfErr`, `dirNFErr`, and `volOffLinErr`.

Discussion

The relevant fields of the parameter block are:

`ref`

On input, a pointer to the first `FSRef` to compare. See `FSRef` (page 837) for a description of the `FSRef` data type.

`parentRef`

On input, a pointer to the second `FSRef` to compare.

You must use `FSCompareFSRefs` (page 476) , or one of the corresponding parameter block functions, `PBCompareFSRefsSync` and `PBCompareFSRefsAsync` (page 586) , to compare `FSRef` structures. It is not possible to compare the `FSRef` structures directly since some bytes may be uninitialized, case-insensitive text, or contain hint information.

Some volume formats may be able to tell that two `FSRef` structures would refer to two different files or directories, without having to actually find those objects. In this case, the volume format may return `errFSRefsDifferent` even if one or both objects no longer exist. Similarly, if the `FSRef` structures are for objects on different volumes, the File Manager will return `diffVolErr` even if one or both volumes are no longer mounted.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBCreateDirectoryUnicodeAsync

Creates a new directory (folder) with a Unicode name.

```
void PBCreateDirectoryUnicodeAsync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 837) for a description of the `FSRefParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information about completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function. See "File Manager Result Codes".

`ref`
: On input, a pointer to an `FSRef` (page 837) for the parent directory where the new directory is to be created.

`nameLength`
: On input, the number of Unicode characters in the new directory's name.

`name`
: On input, a pointer to the Unicode name of the new directory.

`whichInfo`
: On input, a bitmap specifying which catalog information fields to set for the new directory. Specify the values for these fields in the `catInfo` field. If you do not wish to set catalog information for the new directory, specify the constant `kFSCatInfoNone`. See "Catalog Information Bitmap Constants" (page 891) for a description of the bits defined for this field.

`catInfo`
: On input, a pointer to the `FSCatalogInfo` (page 826) structure which specifies the values of the new directory's catalog information fields. Specify which fields to set in the `whichInfo` field. Specify `NULL` if you do not wish to set catalog information for the new directory.

`newRef`
: On output, a pointer to the `FSRef` for the new directory. If you do not want the `FSRef` returned, pass `NULL` on input.

`spec`
: On output, a pointer to the `FSSpec` (page 840) for the new directory. If you do not want the `FSSpec` returned, pass `NULL` on input.

`ioDirID`
: On output, the directory ID of the new directory.

You may optionally set catalog information for the new directory using the `whichInfo` and `catInfo` fields; this is equivalent to calling `FSSetCatalogInfo` (page 540) , or one of the corresponding parameter block functions, `PBSetCatalogInfoSync` (page 753) and `PBSetCatalogInfoAsync` (page 751) , after creating the directory.

If possible, you should set the `textEncodingHint` field of the catalog information structure specified in the `catInfo` field. This will be used by the volume format when converting the Unicode filename to other encodings.

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

Declared In
`Files.h`

PBCreateDirectoryUnicodeSync

Creates a new directory (folder) with a Unicode name.

```
OSErr PBCreateDirectoryUnicodeSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 837) for a description of the `FSRefParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ref`

On input, a pointer to an `FSRef` (page 837) for the parent directory where the new directory is to be created.

`nameLength`

On input, the number of Unicode characters in the new directory's name.

`name`

On input, a pointer to the Unicode name of the new directory.

`whichInfo`

On input, a bitmap specifying which catalog information fields to set for the new directory. Specify the values for these fields in the `catInfo` field. If you do not wish to set catalog information for the new directory, specify the constant `kFSCatInfoNone`. See "Catalog Information Bitmap Constants" (page 891) for a description of the bits defined for this field.

`catInfo`

On input, a pointer to the `FSCatalogInfo` (page 826) structure which specifies the values of the new directory's catalog information fields. Specify which fields to set in the `whichInfo` field. Specify `NULL` if you do not wish to set catalog information for the new directory.

`newRef`

On output, a pointer to the `FSRef` for the new directory. If you do not want the `FSRef` returned, pass `NULL` on input.

`spec`

On output, a pointer to the `FSSpec` (page 840) for the new directory. If you do not want the `FSSpec` returned, pass `NULL` on input.

`ioDirID`

On output, the directory ID of the new directory.

You may optionally set catalog information for the new directory using the `whichInfo` and `catInfo` fields; this is equivalent to calling `FSSetCatalogInfo` (page 540) , or one of the corresponding parameter block functions, `PBSetCatalogInfoSync` (page 753) and `PBSetCatalogInfoAsync` (page 751) , after creating the directory.

If possible, you should set the `textEncodingHint` field of the catalog information structure specified in the `catInfo` field. This will be used by the volume format when converting the Unicode filename to other encodings.

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

Declared In
`Files.h`

PBCreateFileIDRefAsync

Establishes a file ID reference for a file. (Deprecated in Mac OS X v10.5. Use `FSGetCatalogInfo` (page 494) instead.)

```
OSErr PBCreateFileIDRefAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the `FIDParam` (page 818) variant of the HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
Most applications do not need to use this function. In general, you should track files using alias records, as described in the Alias Manager documentation. The Alias Manager uses file IDs internally as part of its search algorithms for finding the target of an alias record.

Given a volume reference number, filename, and parent directory ID, the `PBCreateFileIDRefAsync` function creates a structure to hold the name and parent directory ID of the specified file. The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function. `PBCreateFileIDRefAsync` returns the result code `fidExists` if a file ID reference already exists for the file.

`ioNamePtr`
On input, a pointer to the file's name.

`ioVRefNum`
On input, a volume reference number for the volume containing the file.

`ioSrcDirID`
On input, the file's parent directory ID.

`ioFileID`
On output, a file ID. If a file ID reference already exists for the file, `PBCreateFileIDRefAsync` supplies the file ID but returns the result code `fidExists`.

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
Files.h

PBCreateFileIDRefSync

Establishes a file ID reference for a file. (Deprecated in Mac OS X v10.5. Use FSGetCatalogInfo (page 494) instead.)

```
OSErr PBCreateFileIDRefSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the FIDParam (page 818) variant of the HFS parameter block. See HParamBlockRec (page 857) for a description of the HParamBlockRec data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).PBCreateFileIDRefSync returns the result code fidExists if a file ID reference already exists for the file.

Discussion
Most applications do not need to use this function. In general, you should track files using alias records, as described in the Alias Manager documentation. The Alias Manager uses file IDs internally as part of its search algorithms for finding the target of an alias record.

Given a volume reference number, filename, and parent directory ID, the PBCreateFileIDRefSync function creates a structure to hold the name and parent directory ID of the specified file. The relevant fields of the parameter block are:

ioNamePtr
On input, a pointer to the file's name.

ioVRefNum
On input, a volume reference number for the volume containing the file.

ioSrcDirID
On input, the file's parent directory ID.

ioFileID
On output, a file ID. If a file ID reference already exists for the file, PBCreateFileIDRefSync supplies the file ID but returns the result code fidExists.

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
Files.h

PBCreateFileUnicodeAsync

Creates a new file with a Unicode name.

```
void PBCreateFileUnicodeAsync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 837) for a description of the `FSRefParam` data type.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function. See "File Manager Result Codes".

`ref`

On input, a pointer to an `FSRef` (page 837) for the directory where the file is to be created.

`nameLength`

On input, the number of Unicode characters in the file's name.

`name`

On input, a pointer to the Unicode name of the new file.

`whichInfo`

On input, a bitmap specifying which catalog information fields to set for the new file. Specify the values for these fields in the `catInfo` field. If you do not wish to set catalog information for the new file, pass the constant `kFSCatInfoNone` here. See "Catalog Information Bitmap Constants" (page 891) for a description of the bits defined for this field.

`catInfo`

On input, a pointer to the `FSCatalogInfo` (page 826) structure which specifies the values of the new file's catalog information fields. Specify which fields to set in the `whichInfo` field. This field is optional; specify `NULL` if you do not wish to set catalog information for the new file.

`newRef`

On output, a pointer to the `FSRef` for the new file. If you do not want the `FSRef` returned, pass `NULL` on input.

`spec`

On output, a pointer to the `FSSpec` for the new file. If you do not want the `FSSpec` (page 840) returned, pass `NULL` on input.

You may optionally set catalog information for the file using the `whichInfo` and `catInfo` fields; this is equivalent to calling `FSSetCatalogInfo` (page 540) , or one of the corresponding parameter block functions, `PBSetCatalogInfoSync` (page 753) and `PBSetCatalogInfoAsync` (page 751) , after creating the file.

If possible, you should set the `textEncodingHint` field of the catalog information structure specified in the `catInfo` field. This will be used by the volume format when converting the Unicode filename to other encodings.

Special Considerations

If the `PBCreateFileUnicodeAsync` function is present, but is not implemented by a particular volume, the File Manager will emulate this function by making the appropriate call to `PBHCreateAsync` (page 676). However, if the function is not directly supported by the volume, you will not be able to use the long Unicode filenames, or other features added with HFS Plus.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBCreateFileUnicodeSync

Creates a new file with a Unicode name.

```
OSErr PBCreateFileUnicodeSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 837) for a description of the `FSRefParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ref`

On input, a pointer to an `FSRef` (page 837) for the directory where the file is to be created.

`nameLength`

On input, the number of Unicode characters in the file's name.

`name`

On input, a pointer to the Unicode name of the new file.

`whichInfo`

On input, a bitmap specifying which catalog information fields to set for the new file. Specify the values for these fields in the `catInfo` field. If you do not wish to set catalog information for the new file, pass the constant `kFSCatInfoNone` here. See "Catalog Information Bitmap Constants" (page 891) for a description of the bits defined for this field.

`catInfo`

On input, a pointer to the `FSCatalogInfo` (page 826) structure which specifies the values of the new file's catalog information fields. Specify which fields to set in the `whichInfo` field. This field is optional; specify `NULL` if you do not wish to set catalog information for the new file.

`newRef`

On output, a pointer to the `FSRef` for the new file. If you do not want the `FSRef` returned, set this field to `NULL` on input.

`spec`

On output, a pointer to the `FSSpec` (page 840) for the new file. If you do not want the `FSSpec` returned, set this field to `NULL` on input.

You may optionally set catalog information for the new file using the `whichInfo` and `catInfo` fields; this is equivalent to calling `FSSetCatalogInfo` (page 540) , or one of the corresponding parameter block functions, `PBSetCatalogInfoSync` (page 753) and `PBSetCatalogInfoAsync` (page 751) , after creating the file.

If possible, you should set the `textEncodingHint` field of the catalog information structure specified in the `catInfo` field. This will be used by the volume format when converting the Unicode filename to other encodings.

Special Considerations

If the `PBCreateFileUnicodeSync` function is present, but is not implemented by a particular volume, the File Manager will emulate this function by making the appropriate call to `PBHCreateSync` (page 677). However, if the function is not directly supported by the volume, you will not be able to use the long Unicode filenames, or other features added with HFS Plus.

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

Declared In
`Files.h`

PBCreateForkAsync

Creates a named fork for a file or directory.

```
void PBCreateForkAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

`paramBlock`

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function. If the named fork already exists, the function returns `errFSForkExists`. If the fork name is syntactically invalid or otherwise unsupported for the given volume, `PBCreateForkAsync` returns `errFSBadForkName` or `errFSNameTooLong`.

`ref`

On input, a pointer to an `FSRef` (page 837) specifying the file or directory.

`forkNameLength`

On input, the length of the Unicode name of the new fork.

`forkName`
On input, a pointer to the Unicode name of the fork.

A newly created fork has zero length (that is, its logical end-of-file is zero). The data and resource forks of a file are automatically created and deleted as needed. This is done for compatibility with older APIs, and because data and resource forks are often handled specially. If a given fork always exists for a given volume format (such as data and resource forks for HFS and HFS Plus, or data forks for most other volume formats), an attempt to create that fork when a zero-length fork already exists should return `noErr`; if a non-empty fork already exists then `errFSForkExists` should be returned.

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

Declared In
`Files.h`

PBCreateForkSync

Creates a named fork for a file or directory.

```
OSErr PBCreateForkSync (
   FSForkIOParam *paramBlock
);
```

Parameters

`paramBlock`
A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943). . If the named fork already exists, the function returns `errFSForkExists`. If the fork name is syntactically invalid or otherwise unsupported for the given volume, `PBCreateForkSync` returns `errFSBadForkName` or `errFSNameTooLong`.

Discussion
The relevant fields of the parameter block are:

`ioResult`
On output, the result code of the function. If the named fork already exists, the function returns `errFSForkExists`. If the fork name is syntactically invalid or otherwise unsupported for the given volume, `PBCreateForkAsync` returns `errFSBadForkName` or `errFSNameTooLong`.

`ref`
On input, a pointer to an `FSRef` (page 837) specifying the file or directory.

`forkNameLength`
On input, the length of the Unicode name of the new fork.

`forkName`
On input, a pointer to the Unicode name of the fork.

A newly created fork has zero length (that is, its logical end-of-file is zero). The data and resource forks of a file are automatically created and deleted as needed. This is done for compatibility with older APIs, and because data and resource forks are often handled specially. If a given fork always exists for a given volume format (such as data and resource forks for HFS and HFS Plus, or data forks for most other volume formats), an attempt to create that fork when a zero-length fork already exists should return `noErr`; if a non-empty fork already exists then `errFSForkExists` should be returned.

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

Declared In
`Files.h`

PBDeleteFileIDRefAsync

Deletes a file ID reference. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
OSErr PBDeleteFileIDRefAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `FIDParam` (page 818) variant of the HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
Most applications do not need to use this function. In general, you should track files using alias records, as described in the Alias Manager documentation. The Alias Manager uses file IDs internally as part of its search algorithms for finding the target of an alias record.

The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to a pathname.

`ioVRefNum`
On input, a volume specification for the volume containing the file.

`ioFileID`
On input, the file ID reference to delete. After it has invalidated a file ID reference, the File Manager can no longer resolve that ID reference to a filename and parent directory ID.

Special Considerations

This function is not implemented in Mac OS X.

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
`Files.h`

PBDeleteFileIDRefSync

Deletes a file ID reference. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
OSErr PBDeleteFileIDRefSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `FIDParam` (page 818) variant of the HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

Most applications do not need to use this function. In general, you should track files using alias records, as described in the Alias Manager documentation. The Alias Manager uses file IDs internally as part of its search algorithms for finding the target of an alias record.

The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to a pathname.

`ioVRefNum`

On input, a volume specification for the volume containing the file.

`ioFileID`

On input, the file ID reference to delete. After it has invalidated a file ID reference, the File Manager can no longer resolve that ID reference to a filename and parent directory ID.

Special Considerations

This function is not implemented in Mac OS X.

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

`Files.h`

PBDeleteForkAsync

Deletes a named fork of a file or directory.

```
void PBDeleteForkAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function. If the named fork does not exist, the function returns `errFSForkNotFound`.

`ref`
On input, a pointer to an `FSRef` (page 837) for the file or directory from which to delete the fork.

`forkNameLength`
On input, the length of the fork's Unicode name.

`forkName`
On input, a pointer to the Unicode name of the fork to delete.

The `permissions`, `forkRefNum`, `positionMode`, and `positionOffset` fields of the parameter block may be modified by this call.

Any storage allocated to the fork is released. If a given fork always exists for a given volume format (such as data and resource forks for HFS and HFS Plus, or data forks for most other volume formats), this is equivalent to setting the logical size of the fork to zero.

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

Declared In
`Files.h`

PBDeleteForkSync

Deletes a named fork from a file or directory.

```
OSErr PBDeleteForkSync (
   FSForkIOParam *paramBlock
);
```

Parameters

`paramBlock`
A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943). If the named fork does not exist, the function returns `errFSForkNotFound`.

Discussion
The relevant fields of the parameter block are:

`ref`
On input, a pointer to an `FSRef` (page 837) for the file or directory from which to delete the fork.

`forkNameLength`
On input, the length of the fork's Unicode name.

`forkName`
On input, a pointer to the Unicode name of the fork to delete.

The `permissions`, `forkRefNum`, `positionMode`, and `positionOffset` fields of the parameter block may be modified by this call.

Any storage allocated to the fork is released. If a given fork always exists for a given volume format (such as data and resource forks for HFS and HFS Plus, or data forks for most other volume formats), this is equivalent to setting the logical size of the fork to zero.

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

Declared In
`Files.h`

PBDeleteObjectAsync

Deletes a file or an empty directory.

```
void PBDeleteObjectAsync (
   FSRefParam *paramBlock
);
```

Parameters

`paramBlock`
A pointer to a file system reference parameter block. See `FSRefParam` (page 837) for a description of the `FSRefParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
A result code. See "File Manager Result Codes" (page 943). If you attempt to delete a folder for which there is an open catalog iterator, this function succeeds and returns `noErr`. Iteration, however, will continue to work until the iterator is closed.

`ref`
On input, a pointer to the `FSRef` (page 837) for the file or directory to be deleted. If the object to be deleted is a directory, it must be empty (it must contain no files or folders).

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

Declared In
`Files.h`

PBDeleteObjectSync

Deletes a file or an empty directory.

```
OSErr PBDeleteObjectSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 837) for a description of the `FSRefParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943). If you attempt to delete a folder for which there is an open catalog iterator, this function succeeds and returns `noErr`. Iteration, however, will continue to work until the iterator is closed.

Discussion

The relevant field of the parameter block is:

`ref`

On input, a pointer to the `FSRef` (page 837) for the file or directory to be deleted. If the object to be deleted is a directory, it must be empty (it must contain no files or folders).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBDirCreateAsync

Creates a new directory. (Deprecated in Mac OS X v10.4. Use `PBCreateDirectoryUnicodeAsync` (page 587) instead.)

```
OSErr PBDirCreateAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `HFileParam` (page 852) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to the name for the new directory.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioDirID`
On input, the parent directory ID. If the parent directory ID is 0 and the volume specified in the `ioVRefNum` field is the default volume, the new directory is placed in the default directory of the volume. If the parent directory ID is 0 and the volume specified in the `ioVRefNum` field is a volume other than the default volume, the new directory is placed in the root directory of the volume. To create a directory at the root of a volume, regardless of whether that volume is the current default volume, pass the constant `fsRtDirID` (2) in this field. On output, the directory ID of the new directory. Note that a directory ID, unlike a volume reference number, is a long integer.

The `PBDirCreateAsync` function is identical to `PBHCreateAsync` (page 676) except that it creates a new directory instead of a file. The date and time of the directory's creation and last modification are set to the current date and time.

To create a directory with a Unicode name, use the function `FSCreateDirectoryUnicode` (page 479) , or one of the corresponding parameter block calls, `PBCreateDirectoryUnicodeSync` (page 589) and `PBCreateDirectoryUnicodeAsync` (page 587).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDirCreateSync

Creates a new directory. (Deprecated in Mac OS X v10.4. Use `PBCreateDirectoryUnicodeSync` (page 589) instead.)

```
OSErr PBDirCreateSync (
   HParmBlkPtr paramBlock
);
```

Parameters
`paramBlock`
A pointer to the `HFileParam` (page 852) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the name for the new directory.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioDirID`
On input, the parent directory ID. If the parent directory ID is 0 and the volume specified in the `ioVRefNum` field is the default volume, the new directory is placed in the default directory of the volume. If the parent directory ID is 0 and the volume specified in the `ioVRefNum` field is a volume other than the default volume, the new directory is placed in the root directory of the volume. To create a directory at the root of a volume, regardless of whether that volume is the current default volume, pass the constant `fsRtDirID` (2) in this field. On output, the directory ID of the new directory. Note that a directory ID, unlike a volume reference number, is a long integer.

The `PBDirCreateSync` function is identical to `PBHCreateSync` (page 677) except that it creates a new directory instead of a file. The date and time of the directory's creation and last modification are set to the current date and time.

To create a directory with a Unicode name, use the function `FSCreateDirectoryUnicode` (page 479) , or one of the corresponding parameter block calls, `PBCreateDirectoryUnicodeSync` (page 589) and `PBCreateDirectoryUnicodeAsync` (page 587).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTAddAPPLAsync

Adds an application to the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTAddAPPLAsync (
   DTPBPtr paramBlock
);
```

Parameters
`paramBlock`
A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block for this function are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function. See "File Manager Result Codes".

`ioNamePtr`
On input, a pointer to the application's name.

`ioDTRefNum`
On input, the desktop database reference number of the desktop database to which you wish to add an application.

`ioTagInfo`
Reserved; on input, this field must be set to 0.

`ioDirID`
On input, the ID of the application's parent directory.

`ioFileCreator`
On input, the application's signature.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTAddAPPLSync

Adds an application to the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTAddAPPLSync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block for this function are:

`ioNamePtr`
On input, a pointer to the application's name.

`ioDTRefNum`
On input, the desktop database reference number of the desktop database to which you wish to add an application.

`ioTagInfo`
Reserved; on input, this field must be set to 0.

`ioDirID`
On input, the ID of the application's parent directory.

`ioFileCreator`
On input, the application's signature.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTAddIconAsync

Adds an icon definition to the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTAddIconAsync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block for this function are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function. See "File Manager Result Codes".

`ioDTRefNum`
On input, the desktop database reference number of the database to which you wish to add an icon.

`ioTagInfo`
Reserved; on input, this field must be set to 0.

`ioDTBuffer`
On input, a pointer to the buffer holding the icon's bitmap.

`ioDTReqCount`
On input, the size in bytes of the buffer that you've allocated for the icon's bitmap. This value depends on the icon type. Be sure to allocate enough storage for the icon data 1024 bytes is the largest amount

required for any icon in System 7. For a description of the values you can use to indicate the icon's size, see "Icon Size Constants" (page 921).

`ioIconType`
On input, the icon type. See "Icon Type Constants" (page 922) for a description of the values you can use in this field.

`ioFileCreator`
On input, the icon's file creator.

`ioFileType`
On input, the icon's file type.

If the database already contains an icon definition for an icon of that type, file type, and file creator, the new definition replaces the old.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTAddIconSync

Adds an icon definition to the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTAddIconSync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block for this function are:

`ioDTRefNum`
On input, the desktop database reference number of the database to which you wish to add an icon.

`ioTagInfo`
Reserved; on input, this field must be set to 0.

`ioDTBuffer`
On input, a pointer to the buffer holding the icon's bitmap.

`ioDTReqCount`
: On input, the size in bytes of the buffer that you've allocated for the icon's bitmap. This value depends on the icon type. Be sure to allocate enough storage for the icon data 1024 bytes is the largest amount required for any icon in System 7 For a description of the values you can use to indicate the icon's size, see "Icon Size Constants" (page 921).

`ioIconType`
: On input, the icon type. See "Icon Type Constants" (page 922) for a description of the values you can use in this field.

`ioFileCreator`
: On input, the icon's file creator.

`ioFileType`
: On input, the icon's file type.

If the database already contains an icon definition for an icon of that type, file type, and file creator, the new definition replaces the old.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTCloseDown

Closes the desktop database, though your application should never do this itself. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTCloseDown (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant field of the parameter block for this function is:

`ioDTRefNum`
: On input, the desktop database reference number.

System software uses the `PBDTCloseDown` function to close the desktop database; your application should never use this function, which is described here only for completeness. The system software closes the database when the volume is unmounted.

`PBDTCloseDown` runs synchronously only, and though it will not close down the desktop databases of remote volumes, it will invalidate all local desktop database reference values for remote desktop databases.

When the `PBDTCloseDown` function closes the database, it frees all resources allocated by `PBDTOpenInform` (page 623) or `PBDTGetPath` (page 622).

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTDeleteAsync

Removes the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTDeleteAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The `PBDTDeleteAsync` function removes the desktop database from a local volume. You can call `PBDTDeleteAsync` only when the database is closed. Your application should not call `PBDTDeleteAsync` unless absolutely necessary.

The relevant fields of the parameter block for this function are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function. See "File Manager Result Codes".

`ioVRefNum`

On input, the volume reference number of the desktop database to remove.

`ioIndex`
Reserved; on input, this field must be set to 0.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTDeleteSync

Removes the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTDeleteSync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The `PBDTDeleteSync` function removes the desktop database from a local volume. You can call `PBDTDeleteSync` only when the database is closed. Your application should not call `PBDTDeleteSync` unless absolutely necessary.

The relevant fields of the parameter block for this function are:

`ioVRefNum`
On input, the volume reference number of the desktop database to remove.

`ioIndex`
Reserved; on input, this field must be set to 0.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Files.h

PBDTFlushAsync

Saves your changes to the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTFlushAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock
A pointer to a desktop database parameter block. See DTPBRec (page 813) for a description of the DTPBRec data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
If your application adds information to or removes information from the desktop database, use the PBDTFlushAsync function to save your changes. The PBDTFlushAsync function writes the contents of the desktop database specified in the ioDTRefNum field to the volume.

The relevant fields of the parameter block for this function are:

ioCompletion
On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 794).

ioResult
On output, the result code of the function. See "File Manager Result Codes".

ioDTRefNum
On input, the desktop database reference number of the desktop database to flush.

You must call PBDTFlushAsync or PBDTFlushSync (page 610) to update the copy of the desktop database stored on the volume if your application has manipulated information in the database using any of the following functions:

- PBDTAddIconSync (page 605)
- PBDTAddIconAsync (page 604)
- PBDTAddAPPLSync (page 603)
- PBDTAddAPPLAsync (page 602)
- PBDTSetCommentSync (page 630)
- PBDTSetCommentAsync (page 629)
- PBDTRemoveAPPLSync (page 625)
- PBDTRemoveAPPLAsync (page 624)
- PBDTRemoveCommentSync (page 627)
- PBDTRemoveCommentAsync (page 626)

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTFlushSync

Saves your changes to the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTFlushSync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

If your application adds information to or removes information from the desktop database, use the `PBDTFlushSync` function to save your changes. The `PBDTFlushSync` function writes the contents of the desktop database specified in the `ioDTRefNum` field to the volume.

The relevant field of the parameter block for this function is:

`ioDTRefNum`

On input, the desktop database reference number of the desktop database to flush.

You must call `PBDTFlushSync` or `PBDTFlushAsync` (page 609) to update the copy of the desktop database stored on the volume if your application has manipulated information in the database using any of the following functions:

- `PBDTAddIconSync` (page 605)
- `PBDTAddIconAsync` (page 604)
- `PBDTAddAPPLSync` (page 603)
- `PBDTAddAPPLAsync` (page 602)
- `PBDTSetCommentSync` (page 630)
- `PBDTSetCommentAsync` (page 629)
- `PBDTRemoveAPPLSync` (page 625)
- `PBDTRemoveAPPLAsync` (page 624)

- PBDTRemoveCommentSync (page 627)
- PBDTRemoveCommentAsync (page 626)

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

Files.h

PBDTGetAPPLAsync

Identifies the application that can open a file with a given creator. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetAPPLAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See DTPBRec (page 813) for a description of the DTPBRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block for this function are:

ioCompletion

On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 794).

ioResult

On output, the result code. See "File Manager Result Codes".

ioNamePtr

On output, a pointer to the application's name.

ioDTRefNum

On input, the desktop database reference number of the desktop database containing the specified application.

ioIndex

On input, an index into the application list.

ioTagInfo

On output, the application's creation date.

`ioFileCreator`
On input, the signature of the application.

`ioAPPLParID`
On output, the application's parent directory.

A single call, with the `ioIndex` field set to 0, finds the application file with the most recent creation date. If you want to retrieve information about all copies of the application with the given signature, start with `ioIndex` set to 1 and increment this value by 1 with each call to `PBDTGetAPPLAsync` until the result code `afpItemNotFound` is returned in the `ioResult` field; when called multiple times in this fashion, `PBDTGetAPPLAsync` returns information about all the application's copies, including the file with the most recent creation date, in arbitrary order.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTGetAPPLSync

Identifies the application that can open a file with a given creator. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetAPPLSync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block for this function are:

`ioNamePtr`
On output, a pointer to the application's name.

`ioDTRefNum`
On input, the desktop database reference number of the desktop database containing the specified application.

`ioIndex`
On input, an index into the application list.

`ioTagInfo`
On output, the application's creation date.

`ioFileCreator`
On input, the signature of the application.

`ioAPPLParID`
On output, the application's parent directory.

A single call, with the `ioIndex` field set to 0, finds the application file with the most recent creation date. If you want to retrieve information about all copies of the application with the given signature, start with `ioIndex` set to 1 and increment this value by 1 with each call to `PBDTGetAPPLSync` until the result code `afpItemNotFound` is returned in the `ioResult` field; when called multiple times in this fashion, `PBDTGetAPPLSync` returns information about all the application's copies, including the file with the most recent creation date, in arbitrary order.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTGetCommentAsync

Retrieves the user comments for a file or directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetCommentAsync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block for this function are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function. See "File Manager Result Codes".

`ioNamePtr`
On input, a pointer to the name of the file or directory for which you want to retrieve comments.

`ioDTRefNum`
On input, the desktop database reference number of the database in which the specified file or directory is found.

`ioDTBuffer`
On input, a pointer to a buffer allocated to hold the comment text. On output, a pointer to the comment text. Allocate a buffer at least 255 bytes in size. The `PBDTGetCommentAsync` function places up to `ioDTReqCount` bytes of the comment into the buffer as a plain text string and places the actual length of the comment in the `ioDTActCount` field.

`ioDTReqCount`
On input, the size of the buffer allocated to hold the comment.

`ioDTActCount`
On output, the comment size.

`ioDirID`
On input, the parent directory of the file or directory.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTGetCommentSync

Retrieves the user comments for a file or directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetCommentSync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block for this function are:

`ioNamePtr`
On input, a pointer to the name of the file or directory for which you want to retrieve comments.

`ioDTRefNum`
: On input, the desktop database reference number of the database in which the specified file or directory is found.

`ioDTBuffer`
: On input, a pointer to a buffer allocated to hold the comment text. On output, a pointer to the comment text. Allocate a buffer at least 255 bytes in size. The `PBDTGetCommentSync` function places up to `ioDTReqCount` bytes of the comment into the buffer as a plain text string and places the actual length of the comment in the `ioDTActCount` field.

`ioDTReqCount`
: On input, the size of the buffer allocated to hold the comment.

`ioDTActCount`
: On output, the comment size.

`ioDirID`
: On input, the parent directory of the file or directory.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTGetIconAsync

Retrieves an icon definition. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetIconAsync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The `PBDTGetIconAsync` function returns the bitmap for an icon that represents a file of a given type and creator. For example, to get the icon for a file of file type `'SFWR'` created by the application with a signature of `'WAVE'`, specify these two values in the `ioFileType` and `ioFileCreator` fields.

The relevant fields of the parameter block for this function are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function. See "File Manager Result Codes".

`ioDTRefNum`
: On input, the desktop database reference number.

`ioTagInfo`
: Reserved; on input, this field must be set to 0.

`ioDTBuffer`
: On input, a pointer to a buffer to hold the icon's data. On return, a pointer to the bitmap returned in the buffer.

`ioDTReqCount`
: On input, the requested size of the icon's bitmap. Pass the size in bytes of the buffer that you've allocated for the icon's bitmap pointed to by the `ioDTBuffer` field; this value depends on the icon type. Be sure to allocate enough storage for the icon data; 1024 bytes is the largest amount required for any icon in System 7. You can use the constants described in "Icon Size Constants" (page 921) to indicate the amount of memory you have provided for the icon's data.

`ioDTActCount`
: On return, the actual size of the icon's bitmap. If this value is larger than the value specified in the `ioDTReqCount` field, only the amount of data allowed by the value in the `ioDTReqCount` field is valid.

`ioIconType`
: On input, the icon type. For a description of the constants which you can use in this field, see "Icon Type Constants" (page 922).

`ioFileCreator`
: On input, the icon's file creator.

`ioFileType`
: On input, the icon's file type.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTGetIconInfoAsync

Retrieves an icon type and the associated file type supported by a given creator in the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetIconInfoAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block for this function are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function. See "File Manager Result Codes".

`ioDTRefNum`

On input, the desktop database reference number.

`ioIndex`

On input, an index into the icon list.

`ioTagInfo`

Reserved; on input, this field must be set to 0.

`ioDTActCount`

On output, the size of the icon's bitmap.

`ioIconType`

On output, the icon type, including the icon size and color depth. For a description of the values which may be returned in this field, see "Icon Type Constants" (page 922). Ignore any values returned in `ioIconType` that are not listed there; they represent special icons and information used only by the Finder.

`ioFileCreator`

On input, the icon's file creator.

`ioFileType`

On output, the icon's file type.

To step through a list of the icon types and file types supported by an application, make repeated calls to `PBDTGetIconInfoAsync`, specifying a creator and an index value in the `ioIndex` field for each call. Set the index to 1 on the first call, and increment it on each subsequent call until the result code `afpItemNotFound` is returned in the `ioResult` field.

To get a list of file types that an application can natively open, you can use the Translation Manager function, `GetFileTypesThatAppCanNativelyOpen`. For a description of this function, see the *Translation Manager Reference* .

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTGetIconInfoSync

Retrieves an icon type and the associated file type supported by a given creator in the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetIconInfoSync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock
A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block for this function are:

`ioDTRefNum`
On input, the desktop database reference number.

`ioIndex`
On input, an index into the icon list.

`ioTagInfo`
Reserved; on input, this field must be set to 0.

`ioDTActCount`
On output, the size of the icon's bitmap.

`ioIconType`
On output, the icon type, including the icon size and color depth. For a description of the values which may be returned in this field, see "Icon Type Constants" (page 922). Ignore any values returned in `ioIconType` that are not listed there; they represent special icons and information used only by the Finder.

`ioFileCreator`
On input, the icon's file creator.

`ioFileType`
On output, the icon's file type.

To step through a list of the icon types and file types supported by an application, make repeated calls to `PBDTGetIconInfoSync`, specifying a creator and an index value in the `ioIndex` field for each call. Set the index to 1 on the first call, and increment it on each subsequent call until the result code `afpItemNotFound` is returned in the `ioResult` field.

To get a list of file types that an application can natively open, you can use the Translation Manager function, `GetFileTypesThatAppCanNativelyOpen`. For a description of this function, see the *Translation Manager Reference* .

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTGetIconSync

Retrieves an icon definition. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetIconSync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The `PBDTGetIconSync` function returns the bitmap for an icon that represents a file of a given type and creator. For example, to get the icon for a file of file type `'SFWR'` created by the application with a signature of `'WAVE'`, specify these two values in the `ioFileType` and `ioFileCreator` fields.

The relevant fields of the parameter block for this function are:

`ioDTRefNum`

On input, the desktop database reference number.

`ioTagInfo`

Reserved; on input, this field must be set to 0.

`ioDTBuffer`

On input, a pointer to a buffer to hold the icon's data. On return, a pointer to the bitmap returned in the buffer.

`ioDTReqCount`

On input, the requested size of the icon's bitmap. Pass the size in bytes of the buffer that you've allocated for the icon's bitmap, pointed to by the `ioDTBuffer` field; this value depends on the icon type. Be sure to allocate enough storage for the icon data; 1024 bytes is the largest amount required for any icon in System 7. You can use the constants described in "Icon Size Constants" (page 921) to indicate the amount of memory you have provided for the icon's data.

`ioDTActCount`
: On output, the actual size of the icon's bitmap. If this value is larger than the value specified in the `ioDTReqCount` field, only the amount of data allowed by `ioDTReqCount` is valid.

`ioIconType`
: On input, the icon type. For a description of the constants which you can use in this field, see "Icon Type Constants" (page 922).

`ioFileCreator`
: On input, the icon's file creator.

`ioFileType`
: On input, the icon's file type.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTGetInfoAsync

Determines information about the location and size of the desktop database on a particular volume. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetInfoAsync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block for this function are:

`ioCompletion`
: On input, a pointer to a completion function. For more information on completion functions, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function. See "File Manager Result Codes".

`ioVRefNum`
: On output, the volume reference number of the volume where the database files are stored.

`ioDTRefNum`
: On input, the desktop database reference number of the database which you wish to obtain information about.

`ioIndex`
: On output, the number of files comprising the desktop database on the volume.

`ioDirID`
: On output, the parent directory ID of the desktop database.

`ioDTLgLen`
: On output, the logical length of the database files (the sum of the logical lengths of the files that constitute the desktop database for a given volume).

`ioDTPyLen`
: On output, the physical length of the database files (the sum of the physical lengths of the files that constitute the desktop database for a given volume).

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTGetInfoSync

Determines information about the location and size of the desktop database on a particular volume. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetInfoSync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block for this function are:

`ioVRefNum`
: On output, the volume reference number of the volume where the database files are stored.

`ioDTRefNum`
: On input, the desktop database reference number of the database which you wish to obtain information about.

`ioIndex`
On output, the number of files comprising the desktop database on the volume.

`ioDirID`
On output, the parent directory ID of the desktop database.

`ioDTLgLen`
On output, the logical length of the database files (the sum of the logical lengths of the files that constitute the desktop database for a given volume).

`ioDTPyLen`
On output, the physical length of the database files (the sum of the physical lengths of the files that constitute the desktop database for a given volume).

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTGetPath

Gets the reference number of the specified desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetPath (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block for this function are:

`ioNamePtr`
On input, a pointer to the name of the volume associated with the desktop database or the full pathname of the desktop database.

`ioVRefNum`
On input, the volume reference number of the volume associated with the desktop database.

`ioDTRefNum`
On output, the desktop database reference number, which represents the access path to the database. You cannot use the desktop reference number as a file reference number in any File Manager functions other than the desktop database functions. If `PBDTGetPath` fails, it sets this field to 0.

If the desktop database is not already open, `PBDTGetPath` opens it and then returns the reference number. If the desktop database doesn't exist, `PBDTGetPath` creates it .

Special Considerations

`PBDTGetPath` allocates memory in the system heap; do not call it at interrupt time.

This function executes synchronously 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
`Files.h`

PBDTOpenInform

Gets the reference number of the specified desktop database, reporting whether the desktop database was empty when it was opened. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTOpenInform (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock
A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block for this function are:

`ioNamePtr`
On input, a pointer to the name of the volume associated with the desktop database or the full pathname of the desktop database.

`ioVRefNum`
On input, the volume reference number of the volume associated with the desktop database.

`ioDTRefNum`
On output, the desktop database reference number, which represents the access path to the database. You cannot use the desktop reference number as a file reference number in any File Manager functions other than the desktop database functions. If `PBDTOpenInform` fails, it sets this field to 0.

`ioTagInfo`
On output, the return flag (in the low bit of this field). If the desktop database was just created in response to `PBDTOpenInform` (and is therefore empty), `PBDTOpenInform` sets the low bit in this field to 0. If the desktop database had been created before you called `PBDTOpenInform`, `PBDTOpenInform` sets the low bit in this field to 1.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

This function executes synchronously 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

`Files.h`

PBDTRemoveAPPLAsync

Removes an application from the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTRemoveAPPLAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943). When called on an HFS CD volume, PBDTRemoveAPPL returns an afpItemNotFound error, instead of the expected volume locked error (wPrErr).

Discussion

The `PBDTRemoveAPPLAsync` function removes the mapping information for an application from the database specified in the `ioDTRefNum` field. You can call `PBDTRemoveAPPLAsync` even if the application is not present on the volume.

The relevant fields of the parameter block for this function are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function. See "File Manager Result Codes".

`ioNamePtr`

On input, a pointer to the application's name.

`ioDTRefNum`

On input, the desktop database reference number of the desktop database containing the application.

`ioDirID`

On input, the application's parent directory.

`ioFileCreator`
On input, the application's signature.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTRemoveAPPLSync

Removes an application from the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTRemoveAPPLSync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The `PBDTRemoveAPPLSync` function removes the mapping information for an application from the database specified in the `ioDTRefNum` field. You can call `PBDTRemoveAPPLSync` even if the application is not present on the volume.

The relevant fields of the parameter block for this function are:

`ioNamePtr`
On input, a pointer to the application's name.

`ioDTRefNum`
On input, the desktop database reference number of the desktop database containing the application.

`ioDirID`
On input, the application's parent directory.

`ioFileCreator`
On input, the application's signature.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTRemoveCommentAsync

Removes a user comment associated with a file or directory from the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTRemoveCommentAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock
A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block for this function are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function. See "File Manager Result Codes".

`ioNamePtr`
On input, a pointer to the filename or directory name.

`ioDTRefNum`
On input, the desktop database reference number of the database in which the specified file or directory is found.

`ioDirID`
On input, the parent directory ID of the file or directory.

You cannot remove a comment if the file or directory it is associated with is not present on the volume. If no comment was stored for the file, `PBDTRemoveCommentAsync` returns an error.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTRemoveCommentSync

Removes a user comment associated with a file or directory from the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTRemoveCommentSync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block for this function are:

`ioNamePtr`

On input, a pointer to the filename or directory name.

`ioDTRefNum`

On input, the desktop database reference number of the database in which the specified file or directory is found.

`ioDirID`

On input, the parent directory ID of the file or directory.

You cannot remove a comment if the file or directory it is associated with is not present on the volume. If no comment was stored for the file, `PBDTRemoveCommentSync` returns an error.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTResetAsync

Removes information from the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTResetAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See DTPBRec (page 813) for a description of the DTPBRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The PBDTResetAsync function removes all icons, application mappings, and comments from the desktop database specified in the ioDTRefNum field. You can call PBDTResetAsync only when the database is open. It remains open after the data is cleared. Your application should not call PBDTResetAsync unless absolutely necessary.

The relevant fields of the parameter block for this function are:

ioCompletion

On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 794).

ioResult

On output, the result code of the function. See "File Manager Result Codes".

ioDTRefNum

On input, the desktop database reference number of the desktop database to clear.

ioIndex

Reserved; on input, this field must be set to 0.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

Files.h

PBDTResetSync

Removes information from the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTResetSync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The `PBDTResetSync` function removes all icons, application mappings, and comments from the desktop database specified in the `ioDTRefNum` field. You can call `PBDTResetSync` only when the database is open. It remains open after the data is cleared. Your application should not call `PBDTResetSync` unless absolutely necessary.

The relevant fields of the parameter block for this function are:

`ioDTRefNum`

On input, the desktop database reference number of the desktop database to clear.

`ioIndex`

Reserved; on input, this field must be set to 0.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTSetCommentAsync

Adds a user comment for a file or a directory to the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTSetCommentAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block for this function are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function. See "File Manager Result Codes".

`ioNamePtr`
: On input, a pointer to the name of the file or directory.

`ioDTRefNum`
: On input, the desktop database reference number for the desktop database to which to add the user comment.

`ioDTBuffer`
: On input, a pointer to the buffer containing the comment text. Put the comment in the buffer as a plain text string.

`ioDTReqCount`
: On input, the length of the buffer (in bytes) containing the comment text. The maximum length of a comment is 200 bytes; longer comments are truncated. Since the comment is a plain text string and not a Pascal string, the File Manager relies on the value in the `ioDTReqCount` field for determining the length of the buffer.

`ioDirID`
: On input, the parent directory ID of the file or directory.

If the specified object already has a comment in the database, the new comment replaces the old.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTSetCommentSync

Adds a user comment for a file or a directory to the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTSetCommentSync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 813) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block for this function are:

`ioNamePtr`

On input, a pointer to the name of the file or directory.

`ioDTRefNum`

On input, the desktop database reference number for the desktop database to which to add the user comment.

`ioDTBuffer`

On input, a pointer to the buffer containing the comment text. Put the comment in the buffer as a plain text string.

`ioDTReqCount`

On input, the length of the buffer containing the comment text, in bytes. The maximum length of a comment is 200 bytes; longer comments are truncated. Since the comment is a plain text string and not a Pascal string, the File Manager relies on the value in the `ioDTReqCount` field for determining the length of the buffer.

`ioDirID`

On input, the parent directory ID of the file or directory.

If the specified object already has a comment in the database, the new comment replaces the old.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBExchangeFilesAsync

Exchanges the data stored in two files on the same volume. (Deprecated in Mac OS X v10.4. Use `PBExchangeObjectsAsync` (page 635) instead.)

```
OSErr PBExchangeFilesAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `FIDParam` (page 818) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`ioNamePtr`

On input, a pointer to the name of the first file to swap.

`ioVRefNum`

On input, a volume reference number, drive number, or 0 for the default volume.

`ioDestNamePtr`

On input, a pointer to the name of the second file to swap.

`ioDestDirID`

On input, the second file's parent directory ID.

`ioSrcDirID`

On input, the first file's parent directory ID.

Typically, you use `PBExchangeFilesAsync` after creating a new file during a safe save. The `PBExchangeFilesAsync` function changes the fields in the catalog entries that record the location of the data and the modification dates. It swaps both the data forks and the resource forks.

The `PBExchangeFilesAsync` function works on either open or closed files. `PBExchangeFilesAsync` swaps the data in two files by changing some of the information in the volume catalog. If either file is open, `PBExchangeFilesAsync` updates any file control blocks associated with the file. Exchanging the contents of two files requires essentially the same access privileges as opening both files for writing.

The following fields in the catalog entries for the files are exchanged:

- `ioFlStBlk`
- `ioFlLgLen`
- `ioFlPyLen`
- `ioFlRStBlk`
- `ioFlRLgLen`
- `ioFlRPyLen`
- `ioFlMdDat`

In the file control blocks, the `fcbFlNum`, `fcbDirID`, and `fcbCName` fields are exchanged.

You should use `PBExchangeFilesAsync` to preserve the file ID when updating an existing file, in case the file is being tracked through its file ID. The `PBExchangeFilesAsync` function does not require that file ID references exist for the files being exchanged.

To exchange the contents of files with named forks other than the data and resource forks, or of files larger than 2 GB, use the `FSExchangeObjects` (page 486) , `PBExchangeObjectsSync` (page 636) , or `PBExchangeObjectsAsync` (page 635) function.

Special Considerations

Your application will have to swap any open reference numbers to the two files because the file's name and parent directory ID are exchanged in the file control blocks.

Because other programs may have access paths open to one or both of the files exchanged, your application should have exclusive read/write access permission (`fsRdWrPerm`) to both files before calling `PBExchangeFilesAsync`. Exclusive read/write access to both files will ensure that `PBExchangeFilesAsync` doesn't affect another application because it prevents other applications from obtaining write access to one or both of the files exchanged.

`PBExchangeFilesAsync` does not respect the file-locked attribute; it will perform the exchange even if one or both of the files are locked. Obtaining exclusive read/write access to both files before calling `PBExchangeFilesAsync` ensures that the files are unlocked because locked files cannot be opened with write access.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBExchangeFilesSync

Exchanges the data stored in two files on the same volume. (Deprecated in Mac OS X v10.4. Use `PBExchangeObjectsSync` (page 636) instead.)

```
OSErr PBExchangeFilesSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `FIDParam` (page 818) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the name of the first file to swap.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioDestNamePtr`
On input, a pointer to the name of the second file to swap.

`ioDestDirID`
On input, the second file's parent directory ID.

`ioSrcDirID`
On input, the first file's parent directory ID.

Typically, you use `PBExchangeFilesSync` after creating a new file during a safe save. The `PBExchangeFilesSync` function changes the fields in the catalog entries that record the location of the data and the modification dates. It swaps both the data forks and the resource forks.

The `PBExchangeFilesSync` function works on either open or closed files. `PBExchangeFilesSync` swaps the data in two files by changing some of the information in the volume catalog. If either file is open, `PBExchangeFilesSync` updates any file control blocks associated with the file. Exchanging the contents of two files requires essentially the same access privileges as opening both files for writing.

The following fields in the catalog entries for the files are exchanged:

- `ioFlStBlk`
- `ioFlLgLen`
- `ioFlPyLen`
- `ioFlRStBlk`
- `ioFlRLgLen`
- `ioFlRPyLen`
- `ioFlMdDat`

In the file control blocks, the `fcbFlNum`, `fcbDirID`, and `fcbCName` fields are exchanged.

You should use `PBExchangeFilesSync` to preserve the file ID when updating an existing file, in case the file is being tracked through its file ID. The `PBExchangeFilesSync` function does not require that file ID references exist for the files being exchanged.

To exchange the contents of files with named forks other than the data and resource forks, or of files larger than 2 GB, use the `FSExchangeObjects` (page 486) , `PBExchangeObjectsSync` (page 636) , or `PBExchangeObjectsAsync` (page 635) function.

Special Considerations

Your application will have to swap any open reference numbers to the two files because the file's name and parent directory ID are exchanged in the file control blocks.

Because other programs may have access paths open to one or both of the files exchanged, your application should have exclusive read/write access permission (`fsRdWrPerm`) to both files before calling `PBExchangeFilesSync`. Exclusive read/write access to both files will ensure that `PBExchangeFilesSync` doesn't affect another application because it prevents other applications from obtaining write access to one or both of the files exchanged.

PBExchangeFilesSync does not respect the file-locked attribute; it will perform the exchange even if one or both of the files are locked. Obtaining exclusive read/write access to both files before calling PBExchangeFilesSync ensures that the files are unlocked because locked files cannot be opened with write access.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Files.h

PBExchangeObjectsAsync

Swaps the contents of two files.

```
void PBExchangeObjectsAsync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock
A pointer to a file system reference parameter block. See FSRefParam (page 837) for a description of the FSRefParam data type.

Discussion
The relevant fields of the parameter block are:

ioCompletion
On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 794).

ioResult
On output, the result code of the function.

ref
On input, a pointer to an FSRef (page 837) for the first file.

parentRef
On input, a pointer to an FSRef for the second file.

The PBExchangeObjectsAsync function allows programs to implement a "safe save" operation by creating and writing a complete new file and swapping the contents. An alias, FSSpec, or FSRef that refers to the old file will now access the new data. The corresponding information in in-memory data structures are also exchanged.

Either or both files may have open access paths. After the exchange, the access path will refer to the opposite file's data (that is, to the same data it originally referred, which is now part of the other file).

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

Declared In
Files.h

PBExchangeObjectsSync

Swaps the contents of two files.

```
OSErr PBExchangeObjectsSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 837) for a description of the `FSRefParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ref`

On input, a pointer to an `FSRef` (page 837) for the first file.

`parentRef`

On input, a pointer to an `FSRef` for the second file.

The `PBExchangeObjectsSync` function allows programs to implement a "safe save" operation by creating and writing a complete new file and swapping the contents. An alias, `FSSpec`, or `FSRef` that refers to the old file will now access the new data. The corresponding information in in-memory data structures are also exchanged.

Either or both files may have open access paths. After the exchange, the access path will refer to the opposite file's data (that is, to the same data it originally referred, which is now part of the other file).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBFlushFileAsync

Writes the contents of a file's access path buffer to the disk. (Deprecated in Mac OS X v10.4. Use `PBFlushForkAsync` (page 638) instead.)

```
OSErr PBFlushFileAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `IOParam` (page 862) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 866) for a description of the `ParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function.

`ioRefNum`
On input, a file reference number for the file to flush.

After writing the contents of the file to the volume, the `PBFlushFileAsync` function updates the file's entry in the volume catalog.

In the event of a system crash, all cached data not yet written to disk is lost. If you have made changes to space that already exists within a file (you are overwriting existing data before the file's end-of-file), you must use `PBFlushFileAsync` to ensure that everything written to the file will be written to disk. If you flush the fork's cached blocks using `PBFlushFileAsync`, the only possible data loss in a system crash will be the file's modification date.

You do not, however, need to use `PBFlushFileAsync` to flush a file fork before it is closed; the file is automatically flushed when it is closed and all cache blocks associated with it are removed from the cache.

`PBFlushFileSync` flushes an open fork's dirty cached blocks, but may not flush catalog information associated with the file. To flush catalog information, call `FlushVol` (page 466) , or one of the related parameter block calls, `PBFlushVolSync` (page 641) and `PBFlushVolAsync` (page 640).

To update a file larger than 2GB, or a named fork other than the data and resource forks, you must use the `FSFlushFork` (page 490) function, or one of the corresponding parameter block calls, `PBFlushForkSync` (page 639) and `PBFlushForkAsync` (page 638).

Special Considerations

Some information stored on the volume won't be correct until `PBFlushVolAsync` is called.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBFlushFileSync

Writes the contents of a file's access path buffer to the disk. (Deprecated in Mac OS X v10.4. Use `PBFlushForkSync` (page 639) instead.)

```
OSErr PBFlushFileSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `IOParam` (page 862) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 866) for a description of the `ParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant field of the parameter block is:

`ioRefNum`

On input, a file reference number for the file to flush.

After writing the contents of the file to the volume, the `PBFlushFileSync` function updates the file's entry in the volume catalog.

In the event of a system crash, all cached data not yet written to disk is lost. If you have made changes to space that already exists within a file (you are overwriting existing data before the file's end-of-file), you must use `PBFlushFileSync` to ensure that everything written to the file will be written to disk. If you flush the fork's cached blocks using `PBFlushFileSync`, the only possible data loss in a system crash will be the file's modification date.

You do not, however, need to use `PBFlushFileSync` to flush a file fork before it is closed; the file is automatically flushed when it is closed and all cache blocks associated with it are removed from the cache.

`PBFlushFileSync` flushes an open fork's dirty cached blocks, but may not flush catalog information associated with the file. To flush catalog information, call `FlushVol` (page 466) , or one of the related parameter block calls, `PBFlushVolSync` (page 641) and `PBFlushVolAsync` (page 640).

To update a file larger than 2GB, or a named fork other than the data and resource forks, you must use the `FSFlushFork` (page 490) function, or one of the corresponding parameter block calls, `PBFlushForkSync` (page 639) and `PBFlushForkAsync` (page 638).

Special Considerations

Some information stored on the volume won't be correct until `PBFlushVolSync` is called.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBFlushForkAsync

Causes all data written to an open fork to be written to disk.

```
void PBFlushForkAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for more information on the `FSForkIOParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`forkRefNum`

On input, the reference number of the fork to flush.

The `PBFlushForkAsync` function causes the actual fork contents to be written to disk, as well as any other volume structures needed to access the fork. On HFS and HFS Plus, this includes the catalog, extents, and attribute B-trees; the volume bitmap; and the volume header and alternate volume header (the MDB and alternate MDB on HFS volumes), as needed.

On volumes that do not support `PBFlushForkAsync` directly, the entire volume is flushed to be sure all volume structures associated with the fork are written to disk.

You do not, need to use `PBFlushForkAsync` to flush a file fork before it is closed; the file is automatically flushed when it is closed and all cache blocks associated with it are removed from the cache.

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

Declared In
`Files.h`

PBFlushForkSync

Causes all data written to an open fork to be written to disk.

```
OSErr PBFlushForkSync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for more information on the `FSForkIOParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant field of the parameter block is:

`forkRefNum`
On input, the reference number of the fork to flush.

The `PBFlushForkSync` function causes the actual fork contents to be written to disk, as well as any other volume structures needed to access the fork. On HFS and HFS Plus, this includes the catalog, extents, and attribute B-trees; the volume bitmap; and the volume header and alternate volume header (the MDB and alternate MDB on HFS volumes), as needed.

On volumes that do not support `PBFlushForkSync` directly, the entire volume is flushed to be sure all volume structures associated with the fork are written to disk.

You do not, need to use `PBFlushForkSync` to flush a file fork before it is closed; the file is automatically flushed when it is closed and all cache blocks associated with it are removed from the cache.

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

Declared In
`Files.h`

PBFlushVolAsync

Writes the contents of the volume buffer and updates information about the volume. (Deprecated in Mac OS X v10.5. Use `PBFlushVolumeAsync` (page 642) instead.)

```
OSErr PBFlushVolAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to the `VolumeParam` (page 873) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 866) for a description of the `ParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to the name of the volume to flush.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`PBFlushVolAsync` flushes all open files on the volume, and then flushes all volume data structures. On the volume specified by `ioNamePtr` or `ioVRefNum`, the `PBFlushVolAsync` function writes descriptive information about the volume, the contents of the associated volume buffer, and all access path buffers for the volume (if they've changed since the last time `PBFlushVolAsync` was called).

The date and time of the last modification to the volume are set when the modification is made, not when the volume is flushed.

To ensure that all changes to a volume are flushed to the volume, use `PBFlushVolAsync`. You do not, however, need to flush a volume before unmounting it, ejecting it, or putting it offline; this is done automatically.

If changes are made to a file that affect the file's end-of-file, the file's name, the file's Finder information, or the file's location on the volume, then you must use `PBFlushVolAsync`, or one of the other two volume flush functions in this section, to ensure that these changes are written to disk.

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
`Files.h`

PBFlushVolSync

Writes the contents of the volume buffer and updates information about the volume. (Deprecated in Mac OS X v10.5. Use `PBFlushVolumeSync` (page 642) instead.)

```
OSErr PBFlushVolSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the `VolumeParam` (page 873) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 866) for a description of the `ParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the name of the volume to flush.

`ioVRefNum`
On input, the volume reference number, drive number, or 0 for the default volume.

`PBFlushVolSync` flushes all open files on the volume, and then flushes all volume data structures. On the volume specified by `ioNamePtr` or `ioVRefNum`, the `PBFlushVolSync` function writes descriptive information about the volume, the contents of the associated volume buffer, and all access path buffers for the volume (if they've changed since the last time `PBFlushVolSync` was called).

The date and time of the last modification to the volume are set when the modification is made, not when the volume is flushed.

To ensure that all changes to a volume are flushed to the volume, use `PBFlushVolSync`. You do not, however, need to flush a volume before unmounting it, ejecting it, or putting it offline; this is done automatically.

If changes are made to a file that affect the file's end-of-file, the file's name, the file's Finder information, or the file's location on the volume, then you must use `PBFlushVolSync`, or one of the other two volume flush functions in this section, to ensure that these changes are written to disk.

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
`Files.h`

PBFlushVolumeAsync

For the specified volume, writes all open and modified files in the current process to permanent storage.

```
OSStatus PBFlushVolumeAsync (
    FSRefParamPtr paramBlock
);
```

Parameters

paramBlock
A parameter block containing the volume reference number of the volume to flush. See `FSRefParam` (page 837).

Return Value
A result code. See "File Manager Result Codes" (page 943).

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

Declared In
`Files.h`

PBFlushVolumeSync

For the specified volume, writes all open and modified files in the current process to permanent storage.

```
OSStatus PBFlushVolumeSync (
    FSRefParamPtr paramBlock
);
```

Parameters

paramBlock
A parameter block containing the volume reference number of the volume to flush. See `FSRefParam` (page 837).

Return Value
A result code. See "File Manager Result Codes" (page 943).

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

Declared In
`Files.h`

PBFSCopyFileAsync

Duplicates a file and optionally renames it.

```
OSStatus PBFSCopyFileAsync (
    FSRefParamPtr paramBlock
);
```

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

Declared In
`Files.h`

PBFSCopyFileSync

Duplicates a file and optionally renames it.

```
OSStatus PBFSCopyFileSync (
    FSRefParamPtr paramBlock
);
```

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

Declared In
`Files.h`

PBGetCatalogInfoAsync

Returns catalog information about a file or directory. You can use this function to map from an `FSRef` to an `FSSpec`.

```
void PBGetCatalogInfoAsync (
   FSRefParam *paramBlock
);
```

Parameters

`paramBlock`

A pointer to a file system reference parameter block. See `FSRefParam` (page 837) fro s description of the `FSRefParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function.

`ref`
: On input, a pointer to an `FSRef` (page 837) specifying the file or directory for which to retrieve information.

`whichInfo`
: On input, a bitmap specifying the catalog information fields to return. If you don't want any catalog information, set `whichInfo` to the constant `kFSCatInfoNone`. See "Catalog Information Bitmap Constants" (page 891) for a description of the bits in this field.

`catInfo`
: On output, a pointer to an `FSCatalogInfo` (page 826) structure containing the information about the file or directory. Only the information specified in the `whichInfo` field is returned. If you don't want any catalog information, pass `NULL` here.

`spec`
: On output, a pointer to the `FSSpec` (page 840) for the file or directory. This output is optional; if you do not wish the `FSSpec` returned, pass `NULL` here.

`parentRef`
: On output, a pointer to the `FSRef` for the object's parent directory. This output is optional; if you do not wish the parent directory returned, pass `NULL` here. If the object specified in the `ref` field is a volume's root directory, then the `FSRef` returned in this field will not be a valid `FSRef`, since the root directory has no parent object.

`outName`
: On output, a pointer to the Unicode name of the file or directory. On input, pass a pointer to an `HFSUniStr255` (page 855) structure if you wish the name returned; otherwise, pass `NULL`.

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

Declared In
`Files.h`

PBGetCatalogInfoBulkAsync

Returns information about one or more objects from a catalog iterator. This function can return information about multiple objects in a single call.

```
void PBGetCatalogInfoBulkAsync (
   FSCatalogBulkParam *paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a catalog information parameter block. See `FSCatalogBulkParam` (page 824) for a description of the `FSCatalogBulkParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function. When all of the iterator's objects have been returned, the call will return `errFSNoMoreItems`.

`iterator`
: On input, the iterator to use. You can obtain a catalog iterator with the function `FSOpenIterator` (page 515) , or with one of the related parameter block calls, `PBOpenIteratorSync` (page 742) and `PBOpenIteratorAsync` (page 741). Currently, the iterator must be created with the `kFSIterateFlat` option. See `FSIterator` (page 835) for a description of the `FSIterator` data type.

`maximumItems`
: On input, the maximum number of items to return for this call.

`actualItems`
: On output, the actual number of items found for this call.

`containerChanged`
: On output, a value indicating whether or not the container's contents have changed since the previous `PBGetCatalogInfoBulkAsync` call. If `true`, the contents have changed. Objects may still be returned, even though the container has changed. If so, note that if the container has changed, then the total set of items returned may be incorrect: some items may be returned multiple times, and some items may not be returned at all.

`whichInfo`
: On input, a bitmap specifying the catalog information fields to return for each item. If you don't wish any catalog information returned, pass the constant `kFSCatInfoNone` in this field. For a description of the bits in this field, see "Catalog Information Bitmap Constants" (page 891).

`catalogInfo`
: On output, a pointer to an array of catalog information structures; one for each returned item. On input, the `catalogInfo` field should point to an array of `maximumItems` catalog information structures. This field is optional; if you do not wish any catalog information returned, pass `NULL` here. See `FSCatalogInfo` (page 826) for a description of the `FSCatalogInfo` data type.

`refs`
: On input, a pointer to an array of `maximumItems FSRef` (page 837) structures. On output, an `FSRef` is filled out for each returned item. This field is optional; if you do not wish any `FSRef` structures returned, pass `NULL` here.

`names`
: On output, a pointer to an array of names; one for each returned item. If you want the Unicode name for each item found, set this field to point to an array of `maximumItems HFSUniStr255` (page 855) structures. Otherwise, set it to `NULL`.

`specs`
: On input, a pointer to an array of `maximumItems FSSpec` structures. On output, an `FSSpec` structure is filled out for each returned item. This field is optional; if you do not wish any `FSSpec` structures returned, pass `NULL` here.

The `PBGetCatalogInfoBulkAsync` call may complete and return `noErr` with fewer than `maximumItems` items returned. This may be due to various reasons related to the internal implementation. In this case, you may continue to make `PBGetCatalogInfoBulkSync` calls using the same iterator.

Before calling this function, you should determine whether it is available, by calling the `Gestalt` function.

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

Declared In
Files.h

PBGetCatalogInfoBulkSync

Returns information about one or more objects from a catalog iterator. This function can return information about multiple objects in a single call.

```
OSErr PBGetCatalogInfoBulkSync (
   FSCatalogBulkParam *paramBlock
);
```

Parameters

paramBlock
A pointer to a catalog information parameter block. See FSCatalogBulkParam (page 824) for a description of the FSCatalogBulkParam data type.

Return Value

A result code. See "File Manager Result Codes" (page 943). When all of the iterator's objects have been returned, the call will return errFSNoMoreItems.

Discussion

The relevant fields of the parameter block are:

iterator
On input, the iterator to use. You can obtain a catalog iterator with the function FSOpenIterator (page 515) , or with one of the related parameter block calls, PBOpenIteratorSync (page 742) and PBOpenIteratorAsync (page 741). Currently, the iterator must be created with the kFSIterateFlat option. See FSIterator (page 835) for a description of the FSIterator data type.

maximumItems
On input, the maximum number of items to return for this call.

actualItems
On output, the actual number of items found for this call.

containerChanged
On output, a value indicating whether or not the container's contents have changed since the previous PBGetCatalogInfoBulkSync call. If true, the contents have changed. Objects may still be returned, even though the container has changed. If so, note that if the container has changed, then the total set of items returned may be incorrect: some items may be returned multiple times, and some items may not be returned at all.

whichInfo
On input, a bitmap specifying the catalog information fields to return for each item. If you don't wish any catalog information returned, pass the constant kFSCatInfoNone in this field. For a description of the bits in this field, see "Catalog Information Bitmap Constants" (page 891).

catalogInfo
On output, a pointer to an array of catalog information structures; one for each returned item. On input, the catalogInfo field should point to an array of maximumItems catalog information structures. This field is optional; if you do not wish any catalog information returned, pass NULL here. See FSCatalogInfo (page 826) for a description of the FSCatalogInfo data type.

refs
: On input, a pointer to an array of maximumItems HFSUniStr255 (page 855) structures. On output, an FSRef is filled out for each returned item. This field is optional; if you do not wish any FSRef structures returned, pass NULL here.

names
: On output, a pointer to an array of names; one for each returned item. If you want the Unicode name for each item found, set this field to point to an array of maximumItems HFSUniStr255 (page 855) structures. Otherwise, set it to NULL.

specs
: On input, a pointer to an array of maximumItems FSSpec structures. On output, an FSSpec structure is filled out for each returned item. This field is optional; if you do not wish any FSSpec structures returned, pass NULL here.

The PBGetCatalogInfoBulkSync call may complete and return noErr with fewer than maximumItems items returned. This may be due to various reasons related to the internal implementation. In this case, you may continue to make PBGetCatalogInfoBulkSync calls using the same iterator.

Before calling this function, you should determine whether it is available, by calling the Gestalt function.

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

Declared In
Files.h

PBGetCatalogInfoSync

Returns catalog information about a file or directory. You can use this function to map from an FSRef to an FSSpec.

```
OSErr PBGetCatalogInfoSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock
: A pointer to a file system reference parameter block. See FSRefParam (page 837) for a description of the FSRefParam data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

ref
: On input, a pointer to an FSRef (page 837) specifying the file or directory for which to retrieve information.

whichInfo
: On input, a bitmap specifying the catalog information fields to return. If you don't want any catalog information, set whichInfo to the constant kFSCatInfoNone. See "Catalog Information Bitmap Constants" (page 891) for a description of the bits in this field.

`catInfo`
: On output, a pointer to an `FSCatalogInfo` (page 826) structure containing the information about the file or directory. Only the information specified in the `whichInfo` field is returned. If you don't want any catalog information, pass `NULL` here.

`spec`
: On output, a pointer to the `FSSpec` (page 840) for the file or directory. This output is optional; if you do not wish the `FSSpec` returned, pass `NULL` here.

`parentRef`
: On output, a pointer to the `FSRef` for the object's parent directory. This output is optional; if you do not wish the parent directory returned, pass `NULL` here. If the object specified in the `ref` field is a volume's root directory, then the `FSRef` returned in this field will not be a valid `FSRef`, since the root directory has no parent object.

`outName`
: On output, a pointer to the Unicode name of the file or directory. On input, pass a pointer to an `HFSUniStr255` (page 855) structure if you wish the name returned; otherwise, pass `NULL`.

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

Declared In
`Files.h`

PBGetCatInfoAsync

Returns catalog information about a file or directory. (Deprecated in Mac OS X v10.4. Use `PBGetCatalogInfoAsync` (page 643) instead.)

```
OSErr PBGetCatInfoAsync (
   CInfoPBPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to an HFS catalog information parameter block. See `CInfoPBRec` (page 802) for a description of the `CInfoPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The `PBGetCatInfoAsync` function returns information about a file or directory, depending on the values you specify in the `ioFDirIndex`, `ioNamePtr`, `ioVRefNum`, and `ioDirID` or `ioDrDirID` fields. If you need to determine whether the information returned is for a file or a directory, you can test bit 4 of the `ioFlAttrib` field; if that bit is set, the information returned describes a directory.

The `PBGetCatInfoAsync` function selects a file or directory according to these rules:

- If the value of `ioFDirIndex` is positive, `ioNamePtr` is not used as an input parameter and `PBGetCatInfoAsync` returns information about the file or directory whose directory index is `ioFDirIndex` in the directory specified by `ioDirID` (or `ioDrDirID`) on the volume specified by `ioVRefNum` (this will be the root directory if `ioVRefNum` is a volume reference number or a drive number and `ioDirID` is 0). If `ioNamePtr` is not `NULL`, then it must point to a `Str31` buffer where the file or directory name will be returned.

- If the value of `ioFDirIndex` is 0, `PBGetCatInfoAsync` returns information about the file or directory specified by `ioNamePtr` in the directory specified by `ioDirID` (or `ioDrDirID`) on the volume specified by `ioVRefNum` (again, this will be the root directory if `ioVRefNum` is a volume reference number or a drive number and `ioDirID` is 0).
- If the value of `ioFDirIndex` is negative, `PBGetCatInfoAsync` ignores the `ioNamePtr` field and returns information about the directory specified in the `ioDrDirID` field. If `ioNamePtr` is not `NULL`, then it must point to a `Str31` buffer where the directory name will be returned.

With files, `PBGetCatInfoAsync` is similar to `PBHGetFInfoAsync` (page 682) but returns some additional information. If the object is a file, the relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to a pathname. On output, the name of the file is returned in this field, if the file is open. If you do not want the name of the file returned, pass `NULL` in this field.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioFRefNum`
On output, a file reference number. If the file is open, the reference number of the first access path found is returned here .

`ioFDirIndex`
On input, a directory index.

`ioFlAttrib`
On output, the file attributes. See "File Attribute Constants" (page 914) for the meaning of the file attributes.

`ioFlFndrInfo`
On output, information used by the Finder.

`ioDirID`
On input, a directory ID. On output, the file ID. You might need to save the value of `ioDirID` before calling `PBGetCatInfoAsync` if you make subsequent calls with the same parameter block.

`ioFlStBlk`
On output, the first allocation block of the data fork.

`ioFlLgLen`
On output, the logical size (the logical end-of-file) of the data fork, in bytes.

`ioFlPyLen`
On output, the physical size (the physical end-of-file) of the data fork, in bytes.

`ioFlRStBlk`
On output, the first allocation block of the resource fork.

`ioFlRLgLen`
On output, the logical size of the resource fork, in bytes.

`ioFlRPyLen`
On output, the physical size of the resource fork, in bytes.

`ioFlCrDat`
: On output, the date and time of the file's creation. Note that file systems other than AFP, HFS and HFS Plus do not generally support creation dates. For file systems which do not support creation dates, the File Manager sets the `ioFlCrDat` field to 0.

`ioFlMdDat`
: On output, the date and time of the file's last modification.

`ioFlBkDat`
: On output, the date and time of the file's last backup. Note that file systems other than AFP, HFS and HFS Plus do not generally support backup dates. For file systems which do not support backup dates, the File Manager sets the `ioFlBkDat` field to 0.

`ioFlXFndrInfo`
: On output, additional information used by the Finder.

`ioFlParID`
: On output, the directory ID of the file's parent directory.

`ioFlClpSiz`
: On output, the file's clump size.

You can also use `PBGetCatInfoAsync` to determine whether a file has a file ID reference. The value of the file ID is returned in the `ioDirID` field. Because that parameter could also represent a directory ID, call `PBResolveFileIDRefAsync` (page 749) to see if the value is a real file ID. If you want to determine whether a file ID reference exists for a file and create one if it doesn't, use `PBCreateFileIDRefAsync` (page 590) , which will either create a file ID or return `fidExists`.

If the object is a directory, the relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function.

`ioNamePtr`
: On input, a pointer to a pathname. On output, a pointer to the directory name.

`ioVRefNum`
: On input, a volume reference number, drive number, or 0 for the default volume.

`ioFDirIndex`
: On input, a directory index.

`ioFlAttrib`
: On output, the directory attributes. See "File Attribute Constants" (page 914) for the meaning of the bits in this field. The bits in this field for directories are read-only. You cannot alter directory attributes by setting these bits using the functions `PBSetCatInfoSync` (page 755) or `PBSetCatInfoAsync` (page 754). Instead, you can call the `PBHSetFLockSync` (page 724) and `PBHRstFLockSync` (page 718) functions to lock and unlock a directory, and the `PBShareSync` (page 769) and `PBUnshareSync` (page 773) functions to enable and disable file sharing on local directories.

`ioACUser`
: On output, the directory access rights. The `PBGetCatInfoAsync` function returns the information in this field only for shared volumes. As a result, you should set this field to 0 before calling `PBGetCatInfoAsync`.`PBGetCatInfoAsync` does not return the blank access privileges bit in this field; to determine whether a directory has blank access privileges, use the

PBHGetDirAccessAsync (page 680) function. See "User Privileges Constants" (page 930) for a description of the constants that may be returned in this field.

ioDrUsrWds
On output, information used by the Finder.

ioDrDirID
On input, if you wish to obtain information about a specific directory, that directory's ID. Otherwise, if the object returned is a directory, this field contains the directory ID on output.

ioDrNmFls
On output, the number of files in the directory.

ioDrCrDat
On output, the date and time of the directory's creation. Note that file systems other than AFP, HFS and HFS Plus do not generally support creation dates. For file systems which do not support creation dates, the File Manager sets the ioDrCrDat field to 0.

ioDrMdDat
On output, the date and time of the directory's last modification.

ioDrBkDat
On output, the date and time of the directory's last backup. Note that file systems other than AFP, HFS and HFS Plus do not generally support backup dates. For file systems which do not support backup dates, the File Manager sets the ioDrBkDat field to 0.

ioDrFndrInfo
On output, additional information used by the Finder.

ioDrParID
On output, the directory ID of the directory's parent directory.

To get information on a file or directory with named forks, or on a file larger than 2GB, use one of the FSGetCatalogInfo (page 494) , PBGetCatalogInfoSync (page 647) , or PBGetCatalogInfoAsync (page 643) 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
Files.h

PBGetCatInfoSync

Returns catalog information about a file or directory. (Deprecated in Mac OS X v10.4. Use PBGetCatalogInfoSync (page 647) instead.)

```
OSErr PBGetCatInfoSync (
   CInfoPBPtr paramBlock
);
```

Parameters

paramBlock
A pointer to an HFS catalog information parameter block. See CInfoPBRec (page 802) for a description of the CInfoPBRec data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The `PBGetCatInfoSync` function returns information about a file or directory, depending on the values you specify in the `ioFDirIndex`, `ioNamePtr`, `ioVRefNum`, and `ioDirID` or `ioDrDirID` fields. If you need to determine whether the information returned is for a file or a directory, you can test bit 4 of the `ioFlAttrib` field; if that bit is set, the information returned describes a directory.

The `PBGetCatInfoSync` function selects a file or directory according to these rules:

- If the value of `ioFDirIndex` is positive, `ioNamePtr` is not used as an input parameter and `PBGetCatInfoSync` returns information about the file or directory whose directory index is `ioFDirIndex` in the directory specified by `ioDirID` (or `ioDrDirID`) on the volume specified by `ioVRefNum` (this will be the root directory if `ioVRefNum` is a volume reference number or a drive number and `ioDirID` is 0). If `ioNamePtr` is not `NULL`, then it must point to a `Str31` buffer where the file or directory name will be returned.
- If the value of `ioFDirIndex` is 0, `PBGetCatInfoSync` returns information about the file or directory specified by `ioNamePtr` in the directory specified by `ioDirID` (or `ioDrDirID`) on the volume specified by `ioVRefNum` (again, this will be the root directory if `ioVRefNum` is a volume reference number or a drive number and ioDirID is 0).
- If the value of `ioFDirIndex` is negative, `PBGetCatInfoSync` ignores the `ioNamePtr` field and returns information about the directory specified in the `ioDrDirID` field. If `ioNamePtr` is not `NULL`, then it must point to a `Str31` buffer where the directory name will be returned.

With files, `PBGetCatInfoSync` is similar to `PBHGetFInfoSync` (page 683) but returns some additional information. If the object is a file, the relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a pathname. On output, the name of the file is returned in this field, if the file is open. If you do not want the name of the file returned, pass `NULL` in this field.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioFRefNum`
On output, a file reference number. If the file is open, the reference number of the first access path found is returned here.

`ioFDirIndex`
On input, a directory index.

`ioFlAttrib`
On output, the file attributes. See "File Attribute Constants" (page 914) for the meaning of the file attributes.

`ioFlFndrInfo`
On output, information used by the Finder.

`ioDirID`
On input, a directory ID. On output, the file ID. You might need to save the value of `ioDirID` before calling `PBGetCatInfoSync` if you make subsequent calls with the same parameter block.

`ioFlStBlk`
On output, the first allocation block of the data fork.

`ioFlLgLen`
On output, the logical size (the logical end-of-file) of the data fork, in bytes.

`ioFlPyLen`
On output, the physical size (the physical end-of-file) of the data fork, in bytes.

`ioFlRStBlk`
On output, the first allocation block of the resource fork.

`ioFlRLgLen`
On output, the logical size of the resource fork, in bytes.

`ioFlRPyLen`
On output, the physical size of the resource fork, in bytes.

`ioFlCrDat`
On output, the date and time of the file's creation. Note that file systems other than AFP, HFS and HFS Plus do not generally support creation dates. For file systems which do not support creation dates, the File Manager sets the `ioFlCrDat` field to 0.

`ioFlMdDat`
On output, the date and time of the file's last modification.

`ioFlBkDat`
On output, the date and time of the file's last backup. Note that file systems other than AFP, HFS and HFS Plus do not generally support backup dates. For file systems which do not support backup dates, the File Manager sets the `ioFlBkDat` field to 0.

`ioFlXFndrInfo`
On output, additional information used by the Finder.

`ioFlParID`
On output, the directory ID of the file's parent directory.

`ioFlClpSiz`
On output, the file's clump size.

You can also use `PBGetCatInfoSync` to determine whether a file has a file ID reference. The value of the file ID is returned in the `ioDirID` field. Because that parameter could also represent a directory ID, call `PBResolveFileIDRefSync` (page 750) to see if the value is a real file ID. If you want to determine whether a file ID reference exists for a file and create one if it doesn't, use `PBCreateFileIDRefSync` (page 591) , which will either create a file ID or return `fidExists`.

If the object is a directory, the relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a pathname. On output, a pointer to the directory's name.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioFDirIndex`
On input, a directory index.

`ioFlAttrib`
On output, the directory attributes. See "File Attribute Constants" (page 914) for the meaning of the bits in this field. The bits in this field for directories are read-only. You cannot alter directory attributes by setting these bits using the functions `PBSetCatInfoSync` (page 755) or `PBSetCatInfoAsync` (page 754). Instead, you can call the `PBHSetFLockSync` (page 724) and `PBHRstFLockSync` (page 718) functions to lock and unlock a directory, and the `PBShareSync` (page 769) and `PBUnshareSync` (page 773) functions to enable and disable file sharing on local directories.

`ioACUser`
: On output, the directory access rights. The `PBGetCatInfoSync` function returns the information in this field only for shared volumes. As a result, you should set this field to 0 before calling `PBGetCatInfoSync`. `PBGetCatInfoSync` does not return the blank access privileges bit in this field; to determine whether a directory has blank access privileges, use the `PBHGetDirAccessSync` (page 681) function. See "User Privileges Constants" (page 930) for a description of the constants that may be returned here.

`ioDrUsrWds`
: On output, information used by the Finder.

`ioDrDirID`
: On input, if you wish to obtain information about a specific directory, that directory's ID. Otherwise, if the object returned is a directory, this field contains the directory ID on output.

`ioDrNmFls`
: On output, the number of files in the directory.

`ioDrCrDat`
: On output, the date and time of the directory's creation. Note that file systems other than AFP, HFS and HFS Plus do not generally support creation dates. For file systems which do not support creation dates, the File Manager sets the `ioDrCrDat` field to 0.

`ioDrMdDat`
: On output, the date and time of the directory's last modification.

`ioDrBkDat`
: On output, the date and time of the directory's last backup. Note that file systems other than AFP, HFS and HFS Plus do not generally support backup dates. For file systems which do not support backup dates, the File Manager sets the `ioDrBkDat` field to 0.

`ioDrFndrInfo`
: On output, additional information used by the Finder.

`ioDrParID`
: On output, the directory ID of the directory's parent directory.

To get information on a file or directory with named forks, or on a file larger than 2GB, use one of the `FSGetCatalogInfo` (page 494) , `PBGetCatalogInfoSync` (page 647) , or `PBGetCatalogInfoAsync` (page 643) 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
`Files.h`

PBGetEOFAsync

Determines the current logical size of an open file. (Deprecated in Mac OS X v10.4. Use `PBGetForkSizeAsync` (page 664) instead.)

```
OSErr PBGetEOFAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the IOParam (page 862) variant of the basic File Manager parameter block. See ParamBlockRec (page 866) for a description of the ParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

ioCompletion

On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 794).

ioResult

On output, the result code of the function.

ioRefNum

On input, a file reference number for the open file.

ioMisc

On output, the logical size (the logical end-of-file) of the given file. Because the ioMisc field is of type Ptr, you'll need to coerce the value to a long integer to interpret the value correctly.

To determine the size of a named fork other than the data or resource forks, or of a fork larger than 2 GB, use the FSGetForkSize (page 499) function, or one of the corresponding parameter block functions, PBGetForkSizeSync (page 665) and PBGetForkSizeAsync (page 664).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

Files.h

PBGetEOFSync

Determines the current logical size of an open file. (Deprecated in Mac OS X v10.4. Use PBGetForkSizeSync (page 665) instead.)

```
OSErr PBGetEOFSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the IOParam (page 862) variant of the basic File Manager parameter block. See ParamBlockRec (page 866) for a description of the ParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioRefNum`
On input, a file reference number for the open file.

`ioMisc`
On output, a pointer to the logical size (the logical end-of-file) of the given file. Because the `ioMisc` field is of type `Ptr`, you'll need to coerce the value to a long integer to interpret the value correctly.

To determine the size of a named fork other than the data or resource forks, or of a fork larger than 2 GB, use the `FSGetForkSize` (page 499) function, or one of the corresponding parameter block functions, `PBGetForkSizeSync` (page 665) and `PBGetForkSizeAsync` (page 664).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBGetFCBInfoAsync

Gets information about an open file from the file control block. (Deprecated in Mac OS X v10.4. Use `PBGetForkCBInfoAsync` (page 660) instead.)

```
OSErr PBGetFCBInfoAsync (
   FCBPBPtr paramBlock
);
```

Parameters
paramBlock
A pointer to a file control block parameter block. See `FCBPBRec` (page 816) for a description of the `FCBPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to a pathname. You should pass a pointer to a `Str31` value if you want the name of the file returned. If you pass `NULL`, no filename is returned. On output, if `PBGetFCBInfoAsync` executes successfully, a pointer to the name of the specified open file.

`ioVRefNum`
: On input, a volume specification. If you specify a valid index number in the `ioFCBIndx` field, the File Manager returns information on the file having that index in the FCB buffer on the volume specified in this field. This field may contain a drive number or volume reference number. If the value of `ioVRefNum` is 0, all open files are indexed; otherwise, only open files on the specified volume are indexed.

`ioRefNum`
: On input, if the `ioFCBIndx` field is 0, the file reference number of the file to get information about. If the value of `ioFCBIndx` is positive, the `ioRefNum` field is ignored on input and contains the file reference number on output.

`ioFCBIndx`
: On input, an index. If the value of `ioFCBIndx` is positive, the File Manager returns information about the file whose index in the FCB buffer is `ioFCBIndx` and that is located on the volume specified in the `ioVRefNum` field. If the value of `ioFCBIndx` is 0, the File Manager returns information about the file whose file reference number is specified by the `ioRefNum` field.

`ioFCBFlNm`
: On output, the file ID.

`ioFCBFlags`
: On output, file status flags. See "FCB Flags" (page 906) for a description of the bits in this field.

`ioFCBStBlk`
: On output, the first allocation block of the file.

`ioFCBEOF`
: On output, the logical size (the logical end-of-file) of the file.

`ioFCBPLen`
: On output, the physical size (the physical end-of-file) of the file.

`ioFCBCrPs`
: On output, the position of the file mark.

`ioFCBVRefNum`
: On output, the volume reference number.

`ioFCBClpSiz`
: On output, the file clump size.

`ioFCBParID`
: On output, the directory ID of the file's parent directory.

To get information about a fork control block, use one of the functions, `FSGetForkCBInfo` (page 497) , `PBGetForkCBInfoSync` (page 661) , or `PBGetForkCBInfoAsync` (page 660).

Special Considerations

On OS X, the value returned by `PBGetFCBInfoAsync` in the `ioFCBPLen` field may differ from the physical file length reported by `FSGetCatalogInfo`, `PBGetCatInfo`, and related functions. When a write causes a file to grow in size, the physical length reported by `FSGetCatalogInfo` and similar calls increases by the clump size, which is a multiple of the allocation block size. However, the physical length returned by `PBGetFCBInfoAsync` changes according to the allocation block size and the file lengths returned by the respective functions get out of sync.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Files.h`

PBGetFCBInfoSync

Gets information about an open file from the file control block. (Deprecated in Mac OS X v10.4. Use `PBGetForkCBInfoSync` (page 661) instead.)

```
OSErr PBGetFCBInfoSync (
   FCBPBPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to a file control block parameter block. See `FCBPBRec` (page 816) for a description of the `FCBPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to a pathname. You should pass a pointer to a `Str31` value if you want the name of the file returned. If you pass `NULL`, no filename is returned. On output, if `PBGetFCBInfoSync` executes successfully, a pointer to the name of the specified open file.

`ioVRefNum`

On input, a volume specification. If you specify a valid index number in the `ioFCBIndx` field, the File Manager returns information on the file having that index in the FCB buffer on the volume specified in this field. This field may contain a drive number or volume reference number. If the value of `ioVRefNum` is 0, all open files are indexed; otherwise, only open files on the specified volume are indexed.

`ioRefNum`

On input, if the ioFCBIndx field is 0, the file reference number of the file to get information about. If the value of `ioFCBIndx` is positive, the `ioRefNum` field is ignored on input and contains the file reference number on output.

`ioFCBIndx`

On input, an index. If the value of `ioFCBIndx` is positive, the File Manager returns information about the file whose index in the FCB buffer is `ioFCBIndx` and that is located on the volume specified in the `ioVRefNum` field. If the value of `ioFCBIndx` is 0, the File Manager returns information about the file whose file reference number is specified by the `ioRefNum` field.

`ioFCBFlNm`

On output, the file ID.

`ioFCBFlags`

On output, file status flags. See "FCB Flags" (page 906) for a description of the bits in this field.

`ioFCBStBlk`

On output, the first allocation block of the file.

`ioFCBEOF`

On output, the logical size (the logical end-of-file) of the file.

`ioFCBPLen`
On output, the physical size (the physical end-of-file) of the file.

`ioFCBCrPs`
On output, the current position of the file mark.

`ioFCBVRefNum`
On output, the volume reference number.

`ioFCBClpSiz`
On output, the file clump size.

`ioFCBParID`
On output, the directory ID of the file's parent directory.

To get information about a fork control block, use one of the functions, `FSGetForkCBInfo` (page 497) , `PBGetForkCBInfoSync` (page 661) , or `PBGetForkCBInfoAsync` (page 660).

Special Considerations

On OS X, the value returned by `PBGetFCBInfoSync` in the `ioFCBPLen` field may differ from the physical file length reported by `FSGetCatalogInfo`, `PBGetCatInfo`, and related functions. When a write causes a file to grow in size, the physical length reported by `FSGetCatalogInfo` and similar calls increases by the clump size, which is a multiple of the allocation block size. However, the physical length returned by `PBGetFCBInfoSync` changes according to the allocation block size and the file lengths returned by the respective functions get out of sync.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBGetForeignPrivsAsync

Determines the native access-control information for a file or directory stored on a volume managed by a foreign file system. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBGetForeignPrivsAsync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented 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
`Files.h`

PBGetForeignPrivsSync

Determines the native access-control information for a file or directory stored on a volume managed by a foreign file system. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBGetForeignPrivsSync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented 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

`Files.h`

PBGetForkCBInfoAsync

Returns information about a specified open fork, or about all open forks.

```
void PBGetForkCBInfoAsync (
   FSForkCBInfoParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork control block parameter block. See `FSForkCBInfoParam` (page 830) for a description of the `FSForkCBInfoParam` data type.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`desiredRefNum`

On input, if you want information on a specific fork, set this field to that fork's reference number. If you pass a non-zero value in this parameter, the function attempts to get information on the fork specified by that reference number the field is unchanged on output. Pass zero in this field to iterate over all open forks; on output, this field contains the fork's reference number. You can limit this iteration to a specific volume with the `volumeRefNum` field.

`volumeRefNum`

On input, the volume to search, when iterating over multiple forks. To iterate over all open forks on a single volume, specify the volume reference number in this field. To iterate over all open forks on all volumes, set this field to the constant `kFSInvalidVolumeRefNum`. This field is ignored if you specify a fork reference number in the `desiredRefNum` parameter. Set `desiredRefNum` to zero if

you wish to iterate over multiple forks. See `FSVolumeRefNum` (page 847) for a description of the `FSVolumeRefNum` data type.

`iterator`

On input, an iterator. If the `desiredRefNum` parameter is 0, the iterator maintains state between calls to `PBGetForkCBInfoAsync`. Set the `iterator` field to 0 before you begin iterating, on the first call to `PBGetForkCBInfoAsync`. On return, the iterator will be updated; pass this updated iterator in the `iterator` field of the next call to `PBGetForkCBInfoAsync` to continue iterating.

`actualRefNum`

On output, the actual reference number of the open fork that was found.

`ref`

On output, a pointer to the `FSRef` (page 837) for the file or directory that contains the fork. This information is optional; if you do not wish to the `FSRef`, set `ref` to `NULL`.

`forkInfo`

On output, a pointer to an `FSForkInfo` (page 832) structure containing information about the open fork. This information is optional; if you do not wish it returned, set `forkInfo` to `NULL`.

`forkName`

On output, a pointer to the name of the fork. This field is optional; if you do not wish the name returned, set `forkName` to `NULL`. See `HFSUniStr255` (page 855) for a description of the `HFSUniStr255` data type.

Carbon applications are no longer guaranteed access to the FCB table. Instead, applications should use `FSGetForkCBInfo` (page 497) , or one of the related parameter block functions, `PBGetForkCBInfoSync` (page 661) and `PBGetForkCBInfoAsync`, to access information about a fork control block.

Special Considerations

Returning the fork information in the `forkInfo` field generally does not require a disk access; returning the information in the `ref` or `forkName` fields may cause disk access for some volume formats.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBGetForkCBInfoSync

Returns information about a specified open fork, or about all open forks.

```
OSErr PBGetForkCBInfoSync (
   FSForkCBInfoParam *paramBlock
);
```

Parameters

`paramBlock`

A pointer to a fork control block parameter block. See `FSForkCBInfoParam` (page 830) for a description of the `FSForkCBInfoParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943). If you are iterating over multiple forks, the function returns `errFSNoMoreItems` if there are no more open forks to return.

Discussion

The relevant fields of the parameter block are:

`desiredRefNum`
: On input, if you want information on a specific fork, set this field to that fork's reference number. If you pass a non-zero value in this parameter, the function attempts to get information on the fork specified by that reference number the field is unchanged on output. Pass zero in this field to iterate over all open forks; on output, this field contains the fork's reference number. You can limit this iteration to a specific volume with the `volumeRefNum` field.

`volumeRefNum`
: On input, the volume to search, when iterating over multiple forks. To iterate over all open forks on a single volume, specify the volume reference number in this field. To iterate over all open forks on all volumes, set this field to the constant `kFSInvalidVolumeRefNum`. This field is ignored if you specify a fork reference number in the `desiredRefNum` parameter. Set `desiredRefNum` to zero if you wish to iterate over multiple forks. See `FSVolumeRefNum` (page 847) for a description of the `FSVolumeRefNum` data type.

`iterator`
: On input, an iterator. If the `desiredRefNum` parameter is 0, the iterator maintains state between calls to `PBGetForkCBInfoSync`. Set the `iterator` field to 0 before you begin iterating, on the first call to `PBGetForkCBInfoSync`. On return, the iterator will be updated; pass this updated iterator in the `iterator` field of the next call to `PBGetForkCBInfoSync` to continue iterating.

`actualRefNum`
: On output, the actual reference number of the open fork that was found.

`ref`
: On output, a pointer to the `FSRef` (page 837) for the file or directory that contains the fork. This information is optional; if you do not wish to the `FSRef`, set `ref` to `NULL`.

`forkInfo`
: On output, a pointer to an `FSForkInfo` (page 832) structure containing information about the open fork. This information is optional; if you do not wish it returned, set `forkInfo` to `NULL`.

`forkName`
: On output, a pointer to the name of the fork. This field is optional; if you do not wish the name returned, set `forkName` to `NULL`. See `HFSUniStr255` (page 855) for a description of the `HFSUniStr255` data type.

Carbon applications are no longer guaranteed access to the FCB table. Instead, applications should use `FSGetForkCBInfo` (page 497) , or one of the related parameter block functions, `PBGetForkCBInfoSync` and `PBGetForkCBInfoAsync` (page 660) , to access information about a fork control block.

Special Considerations

Returning the fork information in the `forkInfo` field generally does not require a disk access; returning the information in the `ref` or `forkName` fields may cause disk access for some volume formats.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBGetForkPositionAsync

Returns the current position of an open fork.

```
void PBGetForkPositionAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam` data type.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`forkRefNum`

On input, the reference number of a fork previously opened by the `FSOpenFork` (page 514) , `PBOpenForkSync` (page 740) , or `PBOpenForkAsync` (page 739) function.

`positionOffset`

On output, the current position of the fork. The returned fork position is relative to the start of the fork (that is, it is an absolute offset in bytes).

Special Considerations

Before calling the `PBGetForkPositionAsync` function, call the `Gestalt` function with the `gestaltFSAttr` selector to determine if `PBGetForkPositionAsync` is available. If the function is available, but is not directly supported by a volume, the File Manager will automatically call `PBGetFPosAsync` (page 666); however, you will not be able to determine the fork position of a named fork other than the data or resource fork, or of a fork larger than 2 GB.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBGetForkPositionSync

Returns the current position of an open fork.

```
OSErr PBGetForkPositionSync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`forkRefNum`
: On input, the reference number of a fork previously opened by the `FSOpenFork` (page 514) , `PBOpenForkSync` (page 740) or `PBOpenForkAsync` (page 739) function.

`positionOffset`
: On output, the current position of the fork. The returned fork position is relative to the start of the fork (that is, it is an absolute offset in bytes).

Special Considerations

Before calling the `PBGetForkPositionSync` function, call the `Gestalt` function with the `gestaltFSAttr` selector to determine if `PBGetForkPositionSync` is available. If the function is available, but is not directly supported by a volume, the File Manager will automatically call `PBGetFPosSync` (page 666); however, you will not be able to determine the fork position of a named fork other than the data or resource fork, or of a fork larger than 2 GB.

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

Declared In
`Files.h`

PBGetForkSizeAsync

Returns the size of an open fork.

```
void PBGetForkSizeAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock
: A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function.

`forkRefNum`
: On input, the reference number of the open fork. You can obtain this fork reference number with the `FSOpenFork` (page 514) function, or with one of the corresponding parameter block calls, `PBOpenForkSync` (page 740) and `PBOpenForkAsync` (page 739).

`positionOffset`
: On output, the logical size (the logical end-of-file) of the fork, in bytes. The size returned is the total number of bytes that can be read from the fork; the amount of space actually allocated on the volume (the physical size) will probably be larger.

Special Considerations

To determine whether the `PBGetForkSizeAsync` function is present, call the `Gestalt` function. If `PBGetForkSizeAsync` is present, but is not directly supported by a volume, the File Manager will call `PBGetEOFAsync` (page 654); however, you will not be able to determine the size of a fork other than the data or resource fork, or of a fork larger than 2 GB.

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

Declared In
`Files.h`

PBGetForkSizeSync

Returns the size of an open fork.

```
OSErr PBGetForkSizeSync (
   FSForkIOParam *paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`forkRefNum`
: On input, the reference number of the open fork. You can obtain this fork reference number with the `FSOpenFork` (page 514) function, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 740) and `PBOpenForkAsync` (page 739).

`positionOffset`
: On output, the logical size (the logical end-of-file) of the fork, in bytes. The size returned is the total number of bytes that can be read from the fork; the amount of space actually allocated on the volume (the physical size) will probably be larger.

Special Considerations

To determine whether the `PBGetForkSizeSync` function is present, call the `Gestalt` function. If `PBGetForkSizeSync` is present, but is not directly supported by a volume, the File Manager will call `PBGetEOFSync` (page 655); however, you will not be able to determine the size of a fork other than the data or resource fork, or of a fork larger than 2 GB.

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

Declared In
Files.h

PBGetFPosAsync

Returns the current position of the file mark. (Deprecated in Mac OS X v10.4. Use PBGetForkPositionAsync (page 663) instead.)

```
OSErr PBGetFPosAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the IOParam (page 862) variant of the basic File Manager parameter block. See ParamBlockRec (page 866) for a description of the ParamBlockRec data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

ioCompletion

On input, a pointer to a completion routine. For more information about completion routines, see IOCompletionProcPtr (page 794).

ioResult

On output, the result code of the function.

ioRefNum

On input, the file reference number of an open file.

ioPosOffset

On output, the current position of the mark. The value returned in ioPosOffset is zero-based. Thus, a call to PBGetFPosAsync returns 0 if you call it when the file mark is positioned at the beginning of the file. The ioReqCount, ioActCount, and ioPosMode fields of the parameter block are all set to 0 on output. To determine the current position of a named fork, or of a fork larger than 2GB, use the FSGetForkPosition (page 499) function, or one of the corresponding parameter block calls, PBGetForkPositionSync (page 663) and PBGetForkPositionAsync (page 663).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Files.h

PBGetFPosSync

Returns the current position of the file mark. (Deprecated in Mac OS X v10.4. Use PBGetForkPositionSync (page 663) instead.)

```
OSErr PBGetFPosSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the IOParam (page 862) variant of the basic File Manager parameter block. See ParamBlockRec (page 866) for a description of the ParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

ioRefNum

On input, the file reference number of an open file.

ioPosOffset

On output, the current position of the mark. The value returned in ioPosOffset is zero-based. Thus, a call to PBGetFPosSync returns 0 if you call it when the file mark is positioned at the beginning of the file.

The ioReqCount, ioActCount, and ioPosMode fields of the parameter block are all set to 0 on output.

To determine the current position of a named fork, or of a fork larger than 2GB, use the FSGetForkPosition (page 499) function, or one of the corresponding parameter block calls, PBGetForkPositionSync (page 663) and PBGetForkPositionAsync (page 663).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

Files.h

PBGetUGEntryAsync

Gets a user or group entry from the list of User and Group names and IDs on the local file server. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBGetUGEntryAsync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented 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
`Files.h`

PBGetUGEntrySync

Gets a user or group entry from the list of User and Group names and IDs on a local file server. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBGetUGEntrySync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented 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
`Files.h`

PBGetVolMountInfo

Retrieves a record containing all the information needed to mount a volume, except for passwords. (Deprecated in Mac OS X v10.5. Use `FSVolumeMount` (page 545) instead.)

```
OSErr PBGetVolMountInfo (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the `IOParam` (page 862) variant of the basic File Manager parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a pathname

`ioVRefNum`
On input, a volume specification. This field can contain a volume reference number, drive number, or 0 for the default volume.

`ioBuffer`
On input, a pointer to a buffer to hold the mounting information. The length of the buffer is specified by the value pointed to by the `ioBuffer` field in a previous call to `PBGetVolMountInfoSize` (page 669). On output, the mounting information for the specified volume. You can later pass this structure

to the PBVolumeMount (page 773) function to mount the volume. The mounting information for an AppleShare volume is stored as an AFP mounting record. The PBGetVolMountInfo function does not return the user password or volume password in the AFPVolMountInfo structure. Your application should solicit these passwords from the user and fill in the structure before attempting to mount the remote volume.

This function allows your application to record the mounting information for a volume and then to mount the volume later. This programmatic mounting function stores the mounting information in a structure called the AFPVolMountInfo (page 797) structure.

Special Considerations

This function executes synchronously. You 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

Files.h

PBGetVolMountInfoSize

Determines how much space to allocate for a volume mounting information structure. (Deprecated in Mac OS X v10.5. Use FSVolumeMount (page 545) instead.)

```
OSErr PBGetVolMountInfoSize (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the IOParam (page 862) variant of the basic File Manager parameter block. See HParamBlockRec (page 857) for a description of the HParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

ioNamePtr

On input, a pointer to a pathname

ioVRefNum

On input, a volume specification. This field can contain a volume reference number, drive number, or 0 for the default volume.

ioBuffer

On input, a pointer to storage for the size information, which is of type Integer (2 bytes). If PBGetVolMountInfoSize returns noErr, that integer contains the size of the volume mounting information structure on output.

You should call this function before you call PBGetVolMountInfo (page 668) , to obtain the size of the volume mounting information for which you must allocate storage. Then call PBGetVolMountInfo to retrieve the actual volume mounting information.

Special Considerations

This function executes synchronously. You 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

`Files.h`

PBGetVolumeInfoAsync

Returns information about a volume.

```
void PBGetVolumeInfoAsync (
   FSVolumeInfoParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a volume information parameter block. See `FSVolumeInfoParam` (page 845) for a description of the `FSVolumeInfoParam` data type.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`ioVRefNum`

On input, the volume whose information is to be returned. For information on a particular volume, pass that volume's reference number and set the `volumeIndex` field to 0. To index through the list of mounted volumes, pass the constant `kFSInvalidVolumeRefNum`.On output, the volume reference number of the volume. This is useful when indexing over all mounted volumes, when you have not specified a particular volume reference number on input.

`volumeIndex`

On input, the index of the desired volume, or 0 to use the volume reference number in the `ioVRefNum` field.

`whichInfo`

On input, a bitmap specifying which volume information fields to return in the `volumeInfo` field. If you don't want the information about the volume returned in the `volumeInfo` field, set `whichInfo` to `kFSVolInfoNone`. See "Volume Information Bitmap Constants" (page 938) for a description of the bits in this field.

`volumeInfo`

On output, a pointer to the volume information, as described by the `FSVolumeInfo` (page 842) data type. If you don't want this output, set this field to `NULL`.

`volumeName`

On output, a pointer to the Unicode name of the volume. If you do not wish the name returned, pass `NULL`. Otherwise, pass a pointer to an `HFSUniStr255` (page 855) structure.

`ref`

On output, a pointer to the `FSRef` (page 837) for the volume's root directory. If you do not wish the root directory returned, pass `NULL`.

You can specify a particular volume or index through the list of mounted volumes. To get information on a particular volume, pass the volume reference number of the desired volume in the `ioVRefNum` field and set the `volumeIndex` field to zero. To index through the list of mounted volumes, pass `kFSInvalidVolumeRefNum` in the `ioVRefNum` field and set `volumeIndex` to the index, starting at 1 with the first call to `PBGetVolumeInfoAsync`.

To get information about the root directory of a volume, use the `FSGetCatalogInfo` (page 494) function, or one of the corresponding parameter block calls, `PBGetCatalogInfoSync` (page 647) and `PBGetCatalogInfoAsync` (page 643).

Special Considerations

After an operation that changes the amount of free space on the volume—such as deleting a file—there may be a delay before a call to `PBGetVolumeInfoAsync` returns the updated amount. This is because the File Manager caches and periodically updates file system information, to reduce the number of calls made to retrieve the information from the file system. Currently, the File Manager updates its information every 15 seconds. This primarily affects NFS volumes. DOS, SMB, UFS and WebDAV volumes were also affected by this in previous versions of Mac OS X, but behave correctly in Mac OS X version 10.3 and later.

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

Declared In
`Files.h`

PBGetVolumeInfoSync

Returns information about a volume.

```
OSErr PBGetVolumeInfoSync (
   FSVolumeInfoParam *paramBlock
);
```

Parameters

`paramBlock`

A pointer to a volume information parameter block. See `FSVolumeInfoParam` (page 845) for a description of the `FSVolumeInfoParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioVRefNum`

On input, the volume whose information is to be returned. For information on a particular volume, pass that volume's reference number and set the `volumeIndex` field to 0. To index through the list of mounted volumes, pass the constant `kFSInvalidVolumeRefNum`.On output, the volume reference

number of the volume. This is useful when indexing over all mounted volumes, when you have not specified a particular volume reference number on input.

`volumeIndex`
On input, the index of the desired volume, or 0 to use the volume reference number in the `ioVRefNum` field.

`whichInfo`
On input, a bitmap specifying which volume information fields to return in the `volumeInfo` field. If you don't want the information about the volume returned in the `volumeInfo` field, set `whichInfo` to `kFSVolInfoNone`. See "Volume Information Bitmap Constants" (page 938) for a description of the bits in this field.

`volumeInfo`
On output, a pointer to the volume information, as described by the `FSVolumeInfo` (page 842) data type. If you don't want this output, set this field to `NULL`.

`volumeName`
On output, a pointer to the Unicode name of the volume. If you do not wish the name returned, pass `NULL`. Otherwise, pass a pointer to an `HFSUniStr255` (page 855) structure.

`ref`
On output, a pointer to the `FSRef` (page 837) for the volume's root directory. If you do not wish the root directory returned, pass `NULL`.

You can specify a particular volume or index through the list of mounted volumes. To get information on a particular volume, pass the volume reference number of the desired volume in the `ioVRefNum` field and set the `volumeIndex` field to zero. To index through the list of mounted volumes, pass `kFSInvalidVolumeRefNum` in the `ioVRefNum` field and set `volumeIndex` to the index, starting at 1 with the first call to `PBGetVolumeInfoSync`.

To get information about the root directory of a volume, use the `FSGetCatalogInfo` (page 494) function, or one of the corresponding parameter block calls, `PBGetCatalogInfoSync` (page 647) and `PBGetCatalogInfoAsync` (page 643).

Special Considerations

After an operation that changes the amount of free space on the volume—such as deleting a file—there may be a delay before a call to `PBGetVolumeInfoSync` returns the updated amount. This is because the File Manager caches and periodically updates file system information, to reduce the number of calls made to retrieve the information from the file system. Currently, the File Manager updates its information every 15 seconds. This primarily affects NFS volumes. DOS, SMB, UFS and WebDAV volumes were also affected by this in previous versions of Mac OS X, but behave correctly in Mac OS X version 10.3 and later.

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

Declared In
`Files.h`

PBGetXCatInfoAsync

Returns the short name (MS-DOS format name) and the ProDOS information for a file or directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBGetXCatInfoAsync (
   XCInfoPBPtr paramBlock
);
```

Special Considerations

This function is not implemented 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

`Files.h`

PBGetXCatInfoSync

Returns the short name (MS-DOS format name) and the ProDOS information for a file or directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBGetXCatInfoSync (
   XCInfoPBPtr paramBlock
);
```

Special Considerations

This function is not implemented 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

`Files.h`

PBHCopyFileAsync

Duplicates a file and optionally renames it. (Deprecated in Mac OS X v10.5. Use `PBFSCopyFileAsync` (page 643) instead.)

```
OSErr PBHCopyFileAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a `CopyParam` (page 806) variant of the HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function.

`ioNamePtr`
: On input, a pointer to the name of the source file.

`ioVRefNum`
: On input, the volume reference number or drive number for the volume containing the source file. Pass 0 for the default volume.

`ioDstVRefNum`
: On input, the reference number or drive number of the destination volume. Pass 0 for the default volume.

`ioNewName`
: On input, a pointer to the partial pathname for the destination directory. If `ioNewName` is `NULL`, the destination directory is the directory having the ID specified in the `ioNewDirID` field.

`ioCopyName`
: On input, a pointer to the file's new name. The string pointed to by this field must be a filename, not a partial pathname. If you do not wish to rename the file, pass `NULL` in this field.

`ioNewDirID`
: On input, if the `ioNewName` field is `NULL`, the directory ID of the destination directory. If `ioNewName` is not `NULL`, the parent directory ID of the destination directory.

`ioDirID`
: On input, the directory ID of the source directory.

This function is especially useful when you want to copy or move files located on a remote volume, because it allows you to forgo transmitting large amounts of data across a network. This function is used internally by the Finder; most applications do not need to use it.

Special Considerations

This is an optional call for AppleShare file servers. Your application should examine the information returned by the `PBHGetVolParmsAsync` (page 694) function to see if the volume supports `PBHCopyFileAsync`. If the `bHasCopyFile` bit is set in the `vMAttrib` field of the `GetVolParmsInfoBuffer` structure, then the volume supports `PBHCopyFileAsync`.

For AppleShare file servers, the source and destination pathnames must indicate the same file server; however, the parameter block may specify different source and destination volumes on that file server. A useful way to tell if two file server volumes are on the same file server is to call the `PBHGetVolParmsAsync` (page 694) function for each volume and compare the server addresses returned. The server opens source files with read/deny write enabled and destination files with write/deny read and write enabled.

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
`Files.h`

PBHCopyFileSync

Duplicates a file and optionally renames it. (Deprecated in Mac OS X v10.5. Use `PBFSCopyFileSync` (page 643) instead.)

```
OSErr PBHCopyFileSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to a `CopyParam` (page 806) variant of the HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to the name of the source file.

`ioVRefNum`

On input, the volume reference number or drive number for the volume containing the source file. Pass 0 for the default volume.

`ioDstVRefNum`

On input, the reference number or drive number of the destination volume. Pass 0 for the default volume.

`ioNewName`

On input, a pointer to the partial pathname for the destination directory. If `ioNewName` is `NULL`, the destination directory is the directory having the ID specified in the `ioNewDirID` field.

`ioCopyName`

On input, a pointer to the file's new name. The string pointed to by this field must be a filename, not a partial pathname. If you do not wish to rename the file, pass `NULL` in this field.

`ioNewDirID`

On input, if the `ioNewName` field is `NULL`, the directory ID of the destination directory. If `ioNewName` is not `NULL`, the parent directory ID of the destination directory.

`ioDirID`

On input, the directory ID of the source directory.

This function is especially useful when you want to copy or move files located on a remote volume, because it allows you to forgo transmitting large amounts of data across a network. This function is used internally by the Finder; most applications do not need to use it.

Special Considerations

This is an optional call for AppleShare file servers. Your application should examine the information returned by the `PBHGetVolParmsSync` (page 695) function to see if the volume supports `PBHCopyFileSync`. If the `bHasCopyFile` bit is set in the `vMAttrib` field of the `GetVolParmsInfoBuffer` structure, then the volume supports `PBHCopyFileSync`.

For AppleShare file servers, the source and destination pathnames must indicate the same file server; however, the parameter block may specify different source and destination volumes on that file server. A useful way to tell if two file server volumes are on the same file server is to call the `PBHGetVolParmsSync` (page 695) function for each volume and compare the server addresses returned. The server opens source files with read/deny write enabled and destination files with write/deny read and write enabled.

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

`Files.h`

PBHCreateAsync

Creates a new file. (Deprecated in Mac OS X v10.4. Use `PBCreateFileUnicodeAsync` (page 591) instead.)

```
OSErr PBHCreateAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to the `HFileParam` (page 852) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion function. For more information on completion functions, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`ioNamePtr`

On input, a pointer to the name for the new file.

`ioVRefNum`

On input, a volume reference number, drive number, or 0 for the default volume.

`ioDirID`

On input, the directory ID of the parent directory of the new file.

`ioFVersNum`
: On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

The `PBHCreateAsync` function creates both forks of the file the new file is unlocked and empty. The date and time of its creation and last modification are set to the current date and time. If the file created isn't temporary (that is, if it will exist after the user quits the application), the application should call `PBHSetFInfoAsync` (page 721) , after the call to `PBHCreateAsync`, to fill in the information needed by the Finder.

Files created using `PBHCreateAsync` are not automatically opened. If you want to write data to the new file, you must first open the file using one of the file access functions, `FSpOpenDF` (page 531) , `HOpenDF` (page 554) , `PBHOpenDFSync` (page 708) or `PBHOpenDFAsync` (page 706).

The resource fork of the new file exists but is empty. You'll need to call one of the Resource Manager procedures `HCreateResFile` or `FSpCreateResFile` to create a resource map in the file before you can open it (by calling one of the Resource Manager functions `HOpenResFile` or `FSpOpenResFile`).

To create a file with a Unicode filename, use the function `FSCreateFileUnicode` (page 481) , or one of the corresponding parameter block calls, `PBCreateFileUnicodeSync` (page 593) and `PBCreateFileUnicodeAsync` (page 591).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHCreateSync

Creates a new file. (Deprecated in Mac OS X v10.4. Use `PBCreateFileUnicodeSync` (page 593) instead.)

```
OSErr PBHCreateSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `HFileParam` (page 852) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
: On input, a pointer to the name for the new file.

`ioVRefNum`
: On input, a volume reference number, drive number, or 0 for the default volume.

`ioDirID`
On input, the directory ID of the parent directory of the new file.

`ioFVersNum`
On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

The `PBHCreateSync` function creates both the data and resource fork of the file the new file is unlocked and empty. The date and time of its creation and last modification are set to the current date and time. If the file created isn't temporary (that is, if it will exist after the user quits the application), the application should call `PBHSetFInfoSync` (page 722) after the call to `PBHCreateSync` to fill in the information needed by the Finder.

Files created using `PBHCreateSync` are not automatically opened. If you want to write data to the new file, you must first open the file using one of the file access functions, `FSpOpenDF` (page 531) , `HOpenDF` (page 554) , `PBHOpenDFSync` (page 708) or `PBHOpenDFAsync` (page 706).

The resource fork of the new file exists but is empty. You'll need to call one of the Resource Manager procedures `HCreateResFile` or `FSpCreateResFile` to create a resource map in the file before you can open it (by calling one of the Resource Manager functions `HOpenResFile` or `FSpOpenResFile`).

To create a file with a Unicode filename, use the function `FSCreateFileUnicode` (page 481) , or one of the corresponding parameter block calls, `PBCreateFileUnicodeSync` (page 593) and `PBCreateFileUnicodeAsync` (page 591).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHDeleteAsync

Deletes a file or directory. (Deprecated in Mac OS X v10.4. Use `PBDeleteObjectAsync` (page 599) instead.)

```
OSErr PBHDeleteAsync (
   HParmBlkPtr paramBlock
);
```

Parameters
`paramBlock`
A pointer to the `HFileParam` (page 852) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function. If you attempt to delete an open file or a non-empty directory, `PBHDeleteAsync` returns the result code `fBsyErr`. `PBHDeleteAsync` also returns `fBsyErr` if you attempt to delete a directory that has an open working directory associated with it.

`ioNamePtr`
: On input, a pointer to the name of the file or directory to delete.

`ioVRefNum`
: On input, a volume reference number, drive number, or 0 for the default volume.

`ioDirID`
: On input, the directory ID of the parent directory of the file or directory to delete.

`ioFVersNum`
: On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

If the specified target is a file, both the data and the resource fork of the file are deleted. In addition, if a file ID reference for the specified file exists, that file ID reference is also removed. A file must be closed before you can delete it. Similarly, you cannot delete a directory unless it's empty.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHDeleteSync

Deletes a file or directory. (Deprecated in Mac OS X v10.4. Use `PBDeleteObjectSync` (page 600) instead.)

```
OSErr PBHDeleteSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `HFileParam` (page 852) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943). If you attempt to delete an open file or a non-empty directory, `PBHDeleteSync` returns the result code `fBsyErr`. `PBHDeleteSync` also returns `fBsyErr` if you attempt to delete a directory that has an open working directory associated with it.

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
: On input, a pointer to the name of the file or directory to delete.

`ioVRefNum`
: On input, a volume reference number, drive number, or 0 for the default volume.

`ioDirID`
: On input, the directory ID of the parent directory of the file or directory to delete.

`ioFVersNum`
: On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

If the specified target is a file, both the data and the resource fork of the file are deleted. In addition, if a file ID reference for the specified file exists, that file ID reference is also removed. A file must be closed before you can delete it. Similarly, you cannot delete a directory unless it's empty.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHGetDirAccessAsync

Returns the access control information for a directory or file. (Deprecated in Mac OS X v10.5. Use `FSGetCatalogInfo` (page 494) instead.)

```
OSErr PBHGetDirAccessAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `AccessParam` (page 795) variant of an HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function.

`ioNamePtr`
: On input, a pointer to a pathname for the directory or file.

`ioVRefNum`
: On input, a volume specification for the volume containing the directory or file. This field can contain a volume reference number, drive number, or 0 for the default volume.

`ioACOwnerID`
: On output, the user ID for the owner of the directory or file.

`ioACGroupID`
On output, the primary group ID of the directory or file.

`ioACAccess`
On output, the access rights for the directory or file. See "File and Folder Access Privilege Constants" (page 910) for more information on these access rights.

`ioDirID`
On input, the directory ID.

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
`Files.h`

PBHGetDirAccessSync

Returns the access control information for a directory or file. (Deprecated in Mac OS X v10.5. Use `FSGetCatalogInfo` (page 494) instead.)

```
OSErr PBHGetDirAccessSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to the `AccessParam` (page 795) variant of an HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a pathname for the directory or file.

`ioVRefNum`
On input, a volume specification for the volume containing the directory or file. This field can contain a volume reference number, drive number, or 0 for the default volume.

`ioACOwnerID`
On output, the user ID for the owner of the directory or file.

`ioACGroupID`
On output, the primary group ID of the directory or file.

`ioACAccess`
On output, the access rights for the directory or file. See "File and Folder Access Privilege Constants" (page 910) for more information on these access rights.

`ioDirID`
On input, the directory ID.

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
`Files.h`

PBHGetFInfoAsync

Obtains information about a file. (Deprecated in Mac OS X v10.4. Use `PBGetCatalogInfoAsync` (page 643) instead.)

```
OSErr PBHGetFInfoAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the `HFileParam` (page 852) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to a pathname. If the value of the `ioFDirIndex` field is negative or 0, `PBHGetFInfoAsync` returns information about the file in the volume specified by the reference number in the `ioVRefNum` field and having the name given here. On output, a pointer to the name of the file, if the file is open. If you do not wish the name returned, pass `NULL` here.

`ioVRefNum`
On input, a volume reference number or drive number for the volume containing the file, or 0 for the default volume.

`ioFRefNum`
On output, the reference number of the first access path found, if the file is open and if the `ioFDirIndex` field is negative or 0; if the `ioFDirIndex` field is positive...

`ioFDirIndex`
On input, a directory index. If this value is positive, the function returns information about the file having the directory index specified here, on the volume specified in the `ioVRefNum` field and in the directory specified in the `ioDirID` field. If this value is negative or 0, the function returns information about the file on the specified volume, having the name pointed to in the `ioNamePtr` field.

`ioFlAttrib`
: On output, the file attributes. See "File Attribute Constants" (page 914) for a description of the file attributes.

`ioFlFndrInfo`
: On output, Finder information about the file. For a description of the `FInfo` structure, see the *Finder Interface Reference* .

`ioDirID`
: On input, the parent directory ID of the file. On output, the file's file ID.

`ioFVersNum`
: On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

`ioFlStBlk`
: On output, the first allocation block of the data fork.

`ioFlLgLen`
: On output, the logical size (the logical end-of-file) of the file's data fork, in bytes.

`ioFlPyLen`
: On output, the physical size (the physical end-of-file) of the file's data fork, in bytes.

`ioFlRStBlk`
: On output, the first allocation block of the resource fork.

`ioFlRLgLen`
: On output, the logical size of the file's resource fork, in bytes.

`ioFlRPyLen`
: On output, the physical size of the file's resource fork, in bytes.

`ioFlCrDat`
: On output, the date and time of the file's creation.

`ioFlMdDat`
: On output, the date and time of the file's last modification.

You should call `PBHGetFInfoAsync` just before `PBHSetFInfoAsync` (page 721) , so that the current information is present in the parameter block.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHGetFInfoSync

Obtains information about a file. (Deprecated in Mac OS X v10.4. Use `PBGetCatalogInfoSync` (page 647) instead.)

```
OSErr PBHGetFInfoSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the HFileParam (page 852) variant of the basic HFS parameter block. See HParamBlockRec (page 857) for a description of the HParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

ioNamePtr

On input, a pointer to a pathname. If the value of the ioFDirIndex field is negative or 0, PBHGetFInfoSync returns information about the file in the volume specified by the reference number in the ioVRefNum field and having the name given here. On output, a pointer to the name of the file, if the file is open. If you do not wish the name returned, pass NULL here.

ioVRefNum

On input, a volume reference number or drive number for the volume containing the file, or 0 for the default volume.

ioFRefNum

On output, the reference number of the first access path found, if the file is open and if the ioFDirIndex field is negative or 0; if the ioFDirIndex field is positive...

ioFDirIndex

On input, a directory index. If this value is positive, the function returns information about the file having the directory index specified here, on the volume specified in the ioVRefNum field and in the directory specified in the ioDirID field. If this value is negative or 0, the function returns information about the file on the specified volume, having the name pointed to in the ioNamePtr field.

ioFlAttrib

On output, the file attributes. See "File Attribute Constants" (page 914) for a description of the file attributes.

ioFlFndrInfo

On output, Finder information about the file. For a description of the FInfo data type, see the *Finder Interface Reference* .

ioDirID

On input, the parent directory ID of the file. On output, the file's file ID.

ioFVersNum

On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

ioFlStBlk

On output, the first allocation block of the data fork.

ioFlLgLen

On output, the logical size (the logical end-of-file) of the file's data fork, in bytes.

ioFlPyLen

On output, the physical size (the physical end-of-file) of the file's data fork, in bytes.

`ioFlRStBlk`
On output, the first allocation block of the resource fork.

`ioFlRLgLen`
On output, the logical size of the resource fork, in bytes.

`ioFlRPyLen`
On output, the physical size of the resource fork, in bytes.

`ioFlCrDat`
On output, the date and time of the file's creation.

`ioFlMdDat`
On output, the date and time of the file's last modification.

You should call `PBHGetFInfoSync` just before `PBHSetFInfoSync` (page 722) , so that the current information is present in the parameter block.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHGetLogInInfoAsync

Determines the login method used to log on to a particular shared volume. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBHGetLogInInfoAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the `ObjParam` (page 865) variant of the HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to a pathname

`ioVRefNum`
: On input, a volume specification for the shared volume. This field can contain a volume reference number, drive number, or 0 for the default volume.

`ioObjType`
: On output, the login method type. See "Authentication Method Constants" (page 888) for the values that are recognized. Values in the range 7–127 are reserved for future use by Apple Computer, Inc. Values in the range 128–255 are available to your application as user-defined values.

`ioObjNamePtr`
: On output, a pointer to the user name used to establish the session. The login user name is returned as a Pascal string. The maximum size of the user name is 31 characters.

Special Considerations

This function is not implemented 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

`Files.h`

PBHGetLogInInfoSync

Determines the login method used to log on to a particular shared volume. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBHGetLogInInfoSync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented 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

`Files.h`

PBHGetVInfoAsync

Gets detailed information about a volume. (Deprecated in Mac OS X v10.4. Use `PBGetVolumeInfoAsync` (page 670) instead.)

```
OSErr PBHGetVInfoAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the HVolumeParam (page 859) variant of the basic HFS parameter block. See HParamBlockRec (page 857) for a description of the HParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

ioCompletion

On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 794).

ioResult

On output, the result code of the function.

ioNamePtr

On input, a pointer to a buffer. If you specify a negative number in the ioVolIndex field, this buffer should hold the name of the volume for which to return information. On output, a pointer to the volume's name. You should pass a pointer to a Str31 value if you want the name returned. If you pass NULL, no volume name is returned.

ioVRefNum

On input, a volume specification for the volume for which to return information. If the ioVolIndex field is negative, the File Manager uses the value in the ioNamePtr field, along with the value specified in the ioVRefNum field, to determine the volume. If the value in ioVolIndex is 0, the File Manager attempts to access the volume using only the value in this field. On output, the volume reference number.

ioVolIndex

On input, an index used for indexing through all mounted volumes. If this value is positive, the File Manager uses it to find the volume for which to return information. For instance, if the value of ioVolIndex is 2, the File Manager attempts to access the second mounted volume in the VCB queue. If ioVolIndex is negative, the File Manager uses the values in the ioNamePtr and ioVRefNum fields to access the requested volume. If ioVolIndex is 0, the File Manager uses only the value in the ioVRefNum field.

ioVCrDate

On output, the date and time of the volume's initialization.

ioVLsMod

On output, the date and time of the volume's last modification.

ioVAtrb

On output, the volume attributes. See "Volume Information Attribute Constants" (page 937) for a description of the volume attributes returned by this function.

ioVNmFls

On output, the number of files in the root directory of the volume. For performance reasons, the Carbon File Manager does not return the number of files in this field; instead, it sets ioVNmFls to 0.To determine the number of files in the root directory of a volume in Carbon, call PBGetCatInfoAsync (page 648) for the root directory. The number of files in the root directory is returned in the ioDrNmFls field.

`ioVBitMap`
On output, the first block of the volume bitmap.

`ioVAllocPtr`
On output, the block at which the search for the next new file allocation should start.

`ioVNmAlBlks`
On output, the number of allocation blocks on the volume.

`ioVAlBlkSiz`
On output, the size of the allocation blocks.

`ioVClpSiz`
On output, the default clump size.

`ioAlBlSt`
On output, the first block in the volume block map.

`ioVNxtCNID`
On output, the next unused catalog node ID.

`ioVFrBlk`
On output, the number of unused allocation blocks.

`ioVSigWord`
On output, the volume signature. For HFS volumes, this is 'BD' for HFS Plus volumes, this is 'H+'.

`ioVDrvInfo`
On output, the drive number. You can determine whether the given volume is online by inspecting the value of this field. For online volumes, the `ioVDrvInfo` field contains the drive number of the drive containing the specified volume and hence is always greater than 0. If the value returned in `ioVDrvInfo` is 0, the volume is either offline or ejected.

Mac OS X does not support drive numbers; in Mac OS X, the File Manager always returns a value of 1 in this field.

`ioVDRefNum`
On output, the driver reference number. You can determine whether the volume is offline or ejected by inspecting the value of this field. If the volume is offline, the value of `ioVDRefNum` is the negative of the drive number (which is cleared when the volume is placed offline; hence the `ioVDrvInfo` field for an offline volume is zero), and is a negative number. If the volume is ejected, the value of `ioVDRefNum` is the drive number itself, and thus is a positive number. For online volumes, `ioVDRefNum` contains a driver reference number; these numbers are always less than 0.

`ioVFSID`
On output, the file system handling this volume.

`ioVBkUp`
On output, the date and time of the volume's last backup.

`ioVSeqNum`
Used internally.

`ioVWrCnt`
On output, the volume write count.

`ioVFilCnt`
On output, the number of files on the volume.

`ioVDirCnt`
On output, the number of directories on the volume.

`ioVFndrInfo`

On output, Finder information for the volume.

You can get information about all the online volumes by making repeated calls to `PBHGetVInfoAsync`, starting with the value of the `ioVolIndex` field set to 1 and incrementing that value until `PBHGetVInfoAsync` returns `nsvErr`.

If you need to obtain information about HFS Plus volumes, you should use the `FSGetVolumeInfo` (page 500) function, or one of the corresponding parameter block calls, `PBGetVolumeInfoSync` (page 671) and `PBGetVolumeInfoAsync` (page 670). The `PBHGetVInfoAsync` function is still supported for HFS Plus volumes, but there is additional information returned by the `FSGetVolumeInfo` function (such as the date and time that the volume was last checked for consistency).

Special Considerations

After an operation that changes the amount of free space on the volume—such as deleting a file—there may be a delay before a call to `PBHGetVInfoAsync` returns the updated amount. This is because the File Manager caches and periodically updates file system information, to reduce the number of calls made to retrieve the information from the file system. Currently, the File Manager updates its information every 15 seconds. This primarily affects NFS volumes. DOS, SMB, UFS and WebDAV volumes were also affected by this in previous versions of Mac OS X, but behave correctly in Mac OS X version 10.3 and later.

If the value of `ioVolIndex` is negative, the File Manager uses `ioNamePtr` and `ioVRefNum` in the standard way to determine the volume. However, because `PBHGetVInfoAsync` returns the volume name in the buffer whose address you passed in `ioNamePtr`, your input pathname will be modified. If you don't want your input pathname modified, make a copy of it and pass the copy to `PBHGetVInfoAsync`.

The volume name returned by `PBHGetVInfoAsync` is not a full pathname to the volume because it does not contain a colon.

For compatibility with older programs, some values returned by `PBHGetVInfoAsync` are not what is stored in the volume's Volume Control Block (VCB). Specifically:

- `ioVNmAlBlks` and `ioVFrBlk` are pinned to values which, when multiplied by `ioVAlBlkSiz`, are always less than 2 Gigabytes.
- `ioVNmAlBlks` may not include the allocation blocks used by the catalog and extents overflow files.
- $4244 is returned in `ioVSigWord` for both HFS and HFS Plus volumes.

For unpinned total and free byte counts, and for the real `ioVSigWord`, use `PBXGetVolInfoAsync` (page 779) instead of `PBHGetVInfoAsync`.

Version Notes

In non-Carbon applications, you may pass a working directory reference in the `ioVRefNum` field; if you pass a working directory reference in that field, the number of files and directories in the specified directory is returned in the `ioVNmFls` field.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBHGetVInfoSync

Gets detailed information about a volume. (Deprecated in Mac OS X v10.4. Use `PBGetVolumeInfoSync` (page 671) instead.)

```
OSErr PBHGetVInfoSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `HVolumeParam` (page 859) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to a buffer. If you specify a negative number in the `ioVolIndex` field, this buffer should hold the name of the volume for which to return information. On output, a pointer to the volume's name. You should pass a pointer to a `Str31` value if you want the name returned. If you pass `NULL`, no volume name is returned.

`ioVRefNum`

On input, a volume reference number or drive number for the volume for which to return information; or 0 for the default volume. If the `ioVolIndex` field is negative, the File Manager uses the value in the `ioNamePtr` field, along with the value specified in the `ioVRefNum` field, to determine the volume. If the value in `ioVolIndex` is 0, the File Manager attempts to access the volume using only the value in this field. On output, the volume reference number.

`ioVolIndex`

On input, an index used for indexing through all mounted volumes. If this value is positive, the File Manager uses it to find the volume for which to return information. For instance, if the value of `ioVolIndex` is 2, the File Manager attempts to access the second mounted volume in the VCB queue. If `ioVolIndex` is negative, the File Manager uses the values in the `ioNamePtr` and `ioVRefNum` fields to access the requested volume. If `ioVolIndex` is 0, the File Manager uses only the value in the `ioVRefNum` field.

`ioVCrDate`

On output, the date and time of the volume's initialization.

`ioVLsMod`

On output, the date and time of the volume's last modification.

`ioVAtrb`

On output, the volume attributes. See "Volume Information Attribute Constants" (page 937) for a description of the volume attributes returned by this function.

`ioVNmFls`

On output, the number of files in the root directory of the volume. For performance reasons, the Carbon File Manager does not return the number of files in this field; instead, it sets `ioVNmFls` to 0.To determine the number of files in the root directory of a volume in Carbon, call `PBGetCatInfoSync` (page 651) for the root directory. The number of files in the root directory is returned in the `ioDrNmFls` field.

`ioVBitMap`
On output, the first block of the volume bitmap.

`ioVAllocPtr`
On output, the block at which the search for the next new file allocation should start.

`ioVNmAlBlks`
On output, the number of allocation blocks on the volume.

`ioVAlBlkSiz`
On output, the size of the allocation blocks.

`ioVClpSiz`
On output, the default clump size.

`ioAlBlSt`
On output, the first block in the volume block map.

`ioVNxtCNID`
On output, the next unused catalog node ID.

`ioVFrBlk`
On output, the number of unused allocation blocks.

`ioVSigWord`
On output, the volume signature. For HFS volumes, this is 'BD' for HFS Plus volumes, this is 'H+'.

`ioVDrvInfo`
On output, the drive number. You can determine whether the given volume is online by inspecting the value of this field. For online volumes, the `ioVDrvInfo` field contains the drive number of the drive containing the specified volume and hence is always greater than 0. If the value returned in `ioVDrvInfo` is 0, the volume is either offline or ejected.

Mac OS X does not support drive numbers; in Mac OS X, the File Manager always returns a value of 1 in this field.

`ioVDRefNum`
On output, the driver reference number. You can determine whether the volume is offline or ejected by inspecting the value of this field. If the volume is offline, the value of `ioVDRefNum` is the negative of the drive number (which is cleared when the volume is placed offline; hence the `ioVDrvInfo` field for an offline volume is zero), and is a negative number. If the volume is ejected, the value of `ioVDRefNum` is the drive number itself, and thus is a positive number. For online volumes, `ioVDRefNum` contains a driver reference number; these numbers are always less than 0.

`ioVFSID`
On output, the file system handling this volume.

`ioVBkUp`
On output, the date and time of the volume's last backup.

`ioVSeqNum`
Used internally.

`ioVWrCnt`
On output, the volume write count.

`ioVFilCnt`
On output, the number of files on the volume.

`ioVDirCnt`
On output, the number of directories on the volume.

`ioVFndrInfo`

On output, Finder information for the volume.

You can get information about all the online volumes by making repeated calls to `PBHGetVInfoSync`, starting with the value of the `ioVolIndex` field set to 1 and incrementing that value until `PBHGetVInfoSync` returns `nsvErr`.

If you need to obtain information about HFS Plus volumes, you should use the `FSGetVolumeInfo` (page 500) function, or one of the corresponding parameter block calls, `PBGetVolumeInfoSync` (page 671) and `PBGetVolumeInfoAsync` (page 670). The `PBHGetVInfoSync` function is still supported for HFS Plus volumes, but there is additional information returned by the `FSGetVolumeInfo` function (such as the date and time that the volume was last checked for consistency).

Special Considerations

After an operation that changes the amount of free space on the volume—such as deleting a file—there may be a delay before a call to `PBHGetVInfoSync` returns the updated amount. This is because the File Manager caches and periodically updates file system information, to reduce the number of calls made to retrieve the information from the file system. Currently, the File Manager updates its information every 15 seconds. This primarily affects NFS volumes. DOS, SMB, UFS and WebDAV volumes were also affected by this in previous versions of Mac OS X, but behave correctly in Mac OS X version 10.3 and later.

If the value of `ioVolIndex` is negative, the File Manager uses `ioNamePtr` and `ioVRefNum` in the standard way to determine the volume. However, because `PBHGetVInfoSync` returns the volume name in the buffer whose address you passed in `ioNamePtr`, your input pathname will be modified. If you don't want your input pathname modified, make a copy of it and pass the copy to `PBHGetVInfoSync`.

The volume name returned by `PBHGetVInfoSync` is not a full pathname to the volume because it does not contain a colon.

For compatibility with older programs, some values returned by `PBHGetVInfoSync` are not what is stored in the volume's Volume Control Block (VCB). Specifically:

- `ioVNmAlBlks` and `ioVFrBlk` are pinned to values which, when multiplied by `ioVAlBlkSiz`, are always less than 2 Gigabytes.
- `ioVNmAlBlks` may not include the allocation blocks used by the catalog and extents overflow files.
- $4244 is returned in `ioVSigWord` for both HFS and HFS Plus volumes.

For unpinned total and free byte counts, and for the real `ioVSigWord`, use `PBXGetVolInfoSync` (page 782) instead of `PBHGetVInfoSync`.

Version Notes

In non-Carbon applications, you may pass a working directory reference in the `ioVRefNum` field; if you pass a working directory reference in that field, the number of files and directories in the specified directory is returned in the `ioVNmFls` field.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBHGetVolAsync

Determines the default volume and default directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBHGetVolAsync (
   WDPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a working directory parameter block. See `WDPBRec` (page 877) for a description of the `WDPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The `PBHGetVolAsync` function returns the default volume and directory last set by a call to `HSetVol` (page 559) or `PBHSetVolSync` (page 726). The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`ioNamePtr`

On output, a pointer to the default volume's name. You should pass a pointer to a `Str31` value if you want that name returned. If you pass `NULL` in this field, no volume name is returned.

`ioVRefNum`

On output, the volume reference number of the default volume.

`ioWDProcID`

On output, the working directory user identifier.

`ioWDVRefNum`

On output, the volume reference number of the volume on which the default directory exists.

`ioWDDirID`

On output, the directory ID of the default directory.

Version Notes

When CarbonLib is not present, the `PBHGetVolAsync` function returns a working directory reference number in the `ioVRefNum` parameter if the previous call to `HSetVol` (page 559) (or one of the corresponding parameter block calls) passed in a working directory reference number.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBHGetVolParmsAsync

Returns information about the characteristics of a volume. (Deprecated in Mac OS X v10.5. Use `FSGetVolumeParms` (page 503) instead.)

```
OSErr PBHGetVolParmsAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `HIOParam` (page 855) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`ioNamePtr`

On input, a pointer to the volume's name. You can use a either a name or a volume specification to specify the volume. If you use a volume specification to specify the volume, you should set the `ioNamePtr` field to `NULL`.

`ioVRefNum`

On input, a volume specification. You can use a either a name or a volume specification to specify the volume. A volume specification can be a volume reference number, drive number, or 0 for the default volume.

`ioBuffer`

On input, a pointer to a `GetVolParmsInfoBuffer` (page 847) record; you must allocate this memory to hold the returned attributes. On return, the `PBHGetVolParmsAsync` function places the attributes information in the buffer. Volumes that implement the HFS Plus APIs must use version 3 (or newer) of the `GetVolParmsInfoBuffer` structure. If the version of the `GetVolParmsInfoBuffer` is 2 or less, or the `bSupportsHFSPlusAPIs` bit is clear, then the volume does not implement the HFS Plus APIs and they are being emulated for that volume by the File Manager.

`ioReqCount`

On input, the size, in bytes, of the buffer area pointed to in the `ioBuffer` field.

`ioActCount`

On output, the size of the data actually returned.

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

`Files.h`

PBHGetVolParmsSync

Returns information about the characteristics of a volume. (Deprecated in Mac OS X v10.5. Use `FSGetVolumeParms` (page 503) instead.)

```
OSErr PBHGetVolParmsSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `HIOParam` (page 855) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to the volume's name. You can use a either a name or a volume specification to specify the volume. If you use a volume specification to specify the volume, you should set the `ioNamePtr` field to `NULL`.

`ioVRefNum`

On input, a volume specification. You can use a either a name or a volume specification to specify the volume. A volume specification can be a volume reference number, drive number, or 0 for the default volume.

`ioBuffer`

On input, a pointer to a `GetVolParmsInfoBuffer` (page 847) record; you must allocate this memory to hold the returned attributes. On return, the `PBHGetVolParmsSync` function places the attributes information in the buffer. Volumes that implement the HFS Plus APIs must use version 3 (or newer) of the `GetVolParmsInfoBuffer` structure. If the version of the `GetVolParmsInfoBuffer` is 2 or less, or the `bSupportsHFSPlusAPIs` bit is clear, then the volume does not implement the HFS Plus APIs and they are being emulated for that volume by the File Manager.

`ioReqCount`

On input, the size, in bytes, of the buffer area pointed to in the `ioBuffer` field.

`ioActCount`

On output, the size of the data actually returned.

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

`Files.h`

PBHGetVolSync

Determines the default volume and default directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBHGetVolSync (
   WDPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a working directory parameter block. See `WDPBRec` (page 877) for a description of the `WDPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The `PBHGetVolSync` function returns the default volume and directory last set by a call to `HSetVol` (page 559) or `PBHSetVolSync` (page 726). The relevant fields of the parameter block are:

`ioNamePtr`

On output, a pointer to the default volume's name. Pass a pointer to a `Str31` value if you want that name returned. If you pass `NULL` in this field, no volume name is returned.

`ioVRefNum`

On output, the volume reference number of the default volume.

`ioWDProcID`

On output, the working directory user identifier.

`ioWDVRefNum`

On output, the volume reference number of the volume on which the default directory exists.

`ioWDDirID`

On output, the directory ID of the default directory.

Version Notes

When CarbonLib is not present, the `PBHGetVolSync` function returns a working directory reference number in the `ioVRefNum` parameter if the previous call to `HSetVol` (page 559) (or one of the corresponding parameter block calls) passed in a working directory reference number.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBHMapIDAsync

Determines the name of a user or group given the user or group ID. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
OSErr PBHMapIDAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `ObjParam` (page 865) variant of the HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`ioNamePtr`

On input, a pointer to a pathname.

`ioVRefNum`

On input, a volume specification.

`ioObjType`

On input, the mapping function code its value is 1 if you're mapping a user ID to a user name or 2 if you're mapping a group ID to a group name. See "Mapping Code Constants" (page 926) for more information about the values you can use in this field.

`ioObjNamePtr`

On output, a pointer to the user or group name; the maximum size of the name is 31 characters (preceded by a length byte).

`ioObjID`

On input, the user or group ID to be mapped. AppleShare uses the value 0 to signify Any User.

Special Considerations

See the BSD functions `getpwnam` and `getpwuid`, which correspond to this function on a conceptual level.

Version Notes

Because user and group IDs are interchangeable under AFP 2.1 and later volumes, you might not need to specify a value in the `ioObjType` field.

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

`Files.h`

PBHMapIDSync

Determines the name of a user or group given the user or group ID. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
OSErr PBHMapIDSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the ObjParam (page 865) variant of the HFS parameter block. See HParamBlockRec (page 857) for a description of the HParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

ioNamePtr

On input, a pointer to a pathname.

ioVRefNum

On input, a volume specification.

ioObjType

On input, the mapping function code its value is 1 if you're mapping a user ID to a user name or 2 if you're mapping a group ID to a group name. See "Mapping Code Constants" (page 926) for more information about the values you can use in this field.

ioObjNamePtr

On output, a pointer to the user or group name; the maximum size of the name is 31 characters (preceded by a length byte).

ioObjID

On input, the user or group ID to be mapped. AppleShare uses the value 0 to signify Any User.

Special Considerations

See the BSD functions getpwnam and getpwuid, which correspond to this function on a conceptual level.

Version Notes

Because user and group IDs are interchangeable under AFP 2.1 and later volumes, you might not need to specify a value in the ioObjType field.

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

Files.h

PBHMapNameAsync

Determines the user ID or group ID from a user or group name. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
OSErr PBHMapNameAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `ObjParam` (page 865) variant of the HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`ioNamePtr`

On input, a pointer to a pathname.

`ioVRefNum`

On input, a volume specification.

`ioObjType`

On input, the mapping function code its value is 3 if you're mapping a user name to a user ID or 4 if you're mapping a group name to a group ID. See "Mapping Code Constants" (page 926) for more information on the values you can use in this field.

`ioObjNamePtr`

On input, a pointer to the user or group name. The maximum size of the name is 31 characters. If `NULL` is passed, the ID returned is always 0.

`ioObjID`

On output, the mapped user or group ID.

Special Considerations

See the BSD functions `getpwnam` and `getpwuid`, which correspond to this function on a conceptual level.

Version Notes

Because user and group IDs are interchangeable under AFP 2.1 and later volumes, you might need to set the `ioObjType` field to determine which database (user or group) to search first. If both a user and a group have the same name, this field determines which kind of ID you receive.

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

`Files.h`

PBHMapNameSync

Determines the user ID or group ID from a user or group name. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
OSErr PBHMapNameSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `ObjParam` (page 865) variant of the HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to a pathname.

`ioVRefNum`

On input, a volume specification.

`ioObjType`

On input, the mapping function code its value is 3 if you're mapping a user name to a user ID or 4 if you're mapping a group name to a group ID. See "Mapping Code Constants" (page 926) for more information on the values you can use in this field.

`ioObjNamePtr`

On input, a pointer to the user or group name. The maximum size of the name is 31 characters. If `NULL` is passed, the ID returned is always 0.

`ioObjID`

On output, the mapped user or group ID.

Special Considerations

See the BSD functions `getpwnam` and `getpwuid`, which correspond to this function on a conceptual level.

Version Notes

Because user and group IDs are interchangeable under AFP 2.1 and later volumes, you might need to set the `ioObjType` field to determine which database (user or group) to search first. If both a user and a group have the same name, this field determines which kind of ID you receive.

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

`Files.h`

PBHMoveRenameAsync

Moves a file or directory and optionally renames it. (Deprecated in Mac OS X v10.4. Use `FSMoveObjectAsync` (page 511) instead.)

```
OSErr PBHMoveRenameAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a `CopyParam` (page 806) variant of the HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The `PBHMoveRenameAsync` function allows you to move (not copy) a file or directory. The source and destination pathnames must point to the same file server volume. This function is especially useful when you want to copy or move files located on a remote volume, because it allows you to forgo transmitting large amounts of data across a network. This function is used internally by the Finder; most applications do not need to use it.

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`ioNamePtr`

On input, a pointer to the pathname for the source file or directory.

`ioVRefNum`

On input, a volume reference number or drive number for the volume containing the source file or directory. Pass 0 for the default volume.

`ioNewName`

On input, a pointer to the destination pathname. If `ioNewName` is `NULL`, the destination directory is the directory having the ID specified in the `ioNewDirID` field. If `ioNewName` is not `NULL`, the destination directory is the directory having the partial pathname pointed to by `ioNewName` in the directory having ID `ioNewDirID` on the specified volume.

`ioCopyName`

On input, a pointer to the file's new name. The string pointed to by this field must be a filename, not a partial pathname. If you do not wish to rename the file, pass `NULL` in this field.

`ioNewDirID`

On input, if the `ioNewName` field is `NULL`, the directory ID of the destination directory. If `ioNewName` is not `NULL`, the parent directory ID of the destination directory.

`ioDirID`

On input, the directory ID of the source directory.

Special Considerations

This function is not implemented 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
`Files.h`

PBHMoveRenameSync

Moves a file or directory and optionally renames it. (Deprecated in Mac OS X v10.4. Use `FSMoveObjectSync` (page 512) instead.)

```
OSErr PBHMoveRenameSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a `CopyParam` (page 806) variant of the HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The `PBHMoveRenameSync` function allows you to move (not copy) a file or directory. The source and destination pathnames must point to the same file server volume. This function is especially useful when you want to copy or move files located on a remote volume, because it allows you to forgo transmitting large amounts of data across a network. This function is used internally by the Finder; most applications do not need to use it.

The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the pathname for the source file or directory.

`ioVRefNum`
On input, a volume reference number or drive number for the volume containing the source file or directory. Pass 0 for the default volume.

`ioNewName`
On input, a pointer to the destination pathname. If `ioNewName` is `NULL`, the destination directory is the directory having the ID specified in the `ioNewDirID` field. If `ioNewName` is not `NULL`, the destination directory is the directory having the partial pathname pointed to by `ioNewName` in the directory having ID `ioNewDirID` on the specified volume.

`ioCopyName`
On input, a pointer to the file's new name. The string pointed to by this field must be a filename, not a partial pathname. If you do not wish to rename the file, pass `NULL` in this field.

`ioNewDirID`
On input, if the `ioNewName` field is `NULL`, the directory ID of the destination directory. If `ioNewName` is not `NULL`, the parent directory ID of the destination directory.

`ioDirID`
On input, the directory ID of the source directory.

Special Considerations

This function is not implemented 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

`Files.h`

PBHOpenAsync

Opens the data fork of a file. (Deprecated in Mac OS X v10.4. Use `PBOpenForkAsync` (page 739) instead.)

```
OSErr PBHOpenAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a basic HFS parameter block.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function. If you attempt to open a locked file for writing, `PBHOpenAsync` returns the result code `permErr`. If you request exclusive read/write permission but another access path is already open, `PBHOpenAsync` returns the reference number of the existing access path in `ioRefNum` and `opWrErr` as its function result.

`ioNamePtr`
On input, a pointer to the name of the file.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioRefNum`
On output, a file reference number for accessing the open data fork. If you request exclusive read/write permission but another access path is already open, `PBHOpenAsync` returns the reference number of the existing access path. You should not use this reference number unless your application originally opened the file.

`ioPermssn`
On input, a constant specifying the type of access with which to open the fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 908). You can open

a path for writing even if it accesses a file on a locked volume, and no error is returned until a `PBWriteAsync`, `PBSetEOFAsync` (page 757) , or `PBAllocateAsync` (page 564) call is made.

`ioDirID`
On input, the directory ID of the file's parent directory.

`ioFVersNum`
On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

If you use `PBHOpenAsync` to try to open a file whose name begins with a period, you might mistakenly open a driver instead; subsequent attempts to write data might corrupt data on the target device. To avoid these problems, you should always use `PBHOpenDFAsync` (page 706) instead of `PBHOpenAsync`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHOpenDenyAsync

Opens a file's data fork using the access deny modes. (Deprecated in Mac OS X v10.5. Use `PBOpenForkAsync` (page 739) with deny modes in the permissions field.)

```
OSErr PBHOpenDenyAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a basic HFS parameter block.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function. The function returns the result code `opWrErr` if you've requested write permission and you have already opened the file for writing in that case, the existing file reference number is returned in `ioRefNum`. You should not use this reference number unless your application originally opened the file.

`ioNamePtr`
On input, a pointer to a pathname for the file.

`ioVRefNum`
On input, a volume reference number or drive number for the volume containing the file. Pass 0 to indicate the default volume.

`ioRefNum`
On output, the file reference number for the file.

`ioDenyModes`
On input, the type of access you are requesting to the fork. See "File Access Permission Constants" (page 908) for a description of the types of access that you can request.

`ioDirID`
On input, the parent directory ID of the file.

You should use the `PBHOpenDenyAsync` and `PBHOpenRFDenyAsync` (page 710) functions (or their synchronous counterparts, `PBHOpenDenySync` (page 705) and `PBHOpenRFDenySync` (page 711)) if you want to ensure that you get the access permissions and deny-mode permissions that you request. `PBHOpenDenyAsync` is not retried in any way. If the file cannot be opened because of a deny conflict, the error `afpDenyConflict` is returned and the `ioRefNum` field is set to 0.

You can check that the volume supports AFP deny-mode permissions by checking that the `bHasOpenDeny` bit is set in the `vMAttrib` field returned by the `PBHGetVolParmsSync` (page 695) or `PBHGetVolParmsAsync` (page 694) function. If you don't want to special case volumes that support AFP deny mode permissions, you can use the File Manager permissions. See "File Access Permission Constants" (page 908) for a description of how File Manager permissions are translated to AFP deny-mode permissions.

To open a file's resource fork with access deny permissions, use the `PBHOpenRFDenySync` (page 711) or `PBHOpenRFDenyAsync` (page 710) 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
`Files.h`

PBHOpenDenySync

Opens a file's data fork using the access deny modes. (Deprecated in Mac OS X v10.5. Use `PBOpenForkSync` (page 740) with deny modes in the permissions field.)

```
OSErr PBHOpenDenySync (
   HParmBlkPtr paramBlock
);
```

Parameters
`paramBlock`
A pointer to the `AccessParam` (page 795) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943). The function returns the result code `opWrErr` if you've requested write permission and you have already opened the file for writing in that case, the existing file reference number is returned in `ioRefNum`. You should not use this reference number unless your application originally opened the file.

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a pathname for the file.

`ioVRefNum`
On input, a volume reference number or drive number for the volume containing the file. Pass 0 to indicate the default volume.

`ioRefNum`
On output, the file reference number for the file.

`ioDenyModes`
On input, the type of access you are requesting to the fork. See "File Access Permission Constants" (page 908) for a description of the types of access that you can request.

`ioDirID`
On input, the parent directory ID of the file.

You should use the `PBHOpenDenySync` and `PBHOpenRFDenySync` (page 711) functions (or their asynchronous counterparts, `PBHOpenDenyAsync` (page 704) and `PBHOpenRFDenyAsync` (page 710)) if you want to ensure that you get the access permissions and deny-mode permissions that you request. `PBHOpenDenySync` is not retried in any way. If the file cannot be opened because of a deny conflict, the error `afpDenyConflict` is returned and the `ioRefNum` field is set to 0.

You can check that the volume supports AFP deny-mode permissions by checking that the `bHasOpenDeny` bit is set in the `vMAttrib` field returned by the `PBHGetVolParmsSync` (page 695) or `PBHGetVolParmsAsync` (page 694) function. If you don't want to special case volumes that support AFP deny mode permissions, you can use the File Manager permissions. See "File Access Permission Constants" (page 908) for a description of how File Manager permissions are translated to AFP deny-mode permissions.

To open a file's resource fork with access deny permissions, use the `PBHOpenRFDenySync` (page 711) or `PBHOpenRFDenyAsync` (page 710) 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
`Files.h`

PBHOpenDFAsync

Opens the data fork of a file. (Deprecated in Mac OS X v10.4. Use `PBOpenForkAsync` (page 739) instead.)

```
OSErr PBHOpenDFAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to the `HIOParam` (page 855) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
You should use `PBHOpenDFAsync` instead of the `PBHOpenAsync` (page 703) function; `PBHOpenDFAsync` allows you to safely open a file whose name begins with a period (.).

The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function. If you attempt to open a locked file for writing, `PBHOpenDFAsync` returns the result code `permErr`. If you request exclusive read/write permission but another access path is already open, `PBHOpenDFAsync` returns the reference number of the existing access path in `ioRefNum` and `opWrErr` as its function result.

`ioNamePtr`
On input, a pointer to the name of the file.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioRefNum`
On output, the file reference number for accessing the open data fork. If you request exclusive read/write permission but another access path is already open, `PBHOpenDFAsync` returns the reference number of the existing access path. You should not use this reference number unless your application originally opened the file.

`ioPermssn`
On input, a constant specifying the type of access with which to open the fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 908). You can open a path for writing even if it accesses a file on a locked volume, and no error is returned until a `PBWriteAsync`, `PBSetEOFAsync` (page 757) , or `PBAllocateAsync` (page 564) call is made.

`ioDirID`
On input, the directory ID of the file's parent directory.

`ioFVersNum`
On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

Note that if you wish to access named forks other than the data and resource forks, or forks larger than 2GB, you will need to use the `FSOpenFork` (page 514) function, or one of its corresponding parameter block calls, `PBOpenForkSync` (page 740) or `PBOpenForkAsync` (page 739). If you try to open a fork larger than 2GB with the `PBHOpenDFAsync` function, you will receive an error message.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHOpenDFSync

Opens the data fork of a file. (Deprecated in Mac OS X v10.4. Use `PBOpenForkSync` (page 740) instead.)

```
OSErr PBHOpenDFSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `HIOParam` (page 855) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943). . If you attempt to open a locked file for writing, `PBHOpenDFSync` returns the result code `permErr`. If you request exclusive read/write permission but another access path is already open, `PBHOpenDFSync` returns the reference number of the existing access path in `ioRefNum` and `opWrErr` as its function result.

Discussion

You should use `PBHOpenDFSync` instead of the `PBHOpenSync` (page 714) function; `PBHOpenDFSync` allows you to safely open a file whose name begins with a period (.).

The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to the name of the file.

`ioVRefNum`

On input, a volume reference number, drive number, or 0 for the default volume.

`ioRefNum`

On output, the file reference number for accessing the open data fork. If you request exclusive read/write permission but another access path is already open, `PBHOpenDFSync` returns the reference number of the existing access path. You should not use this reference number unless your application originally opened the file.

`ioPermssn`

On input, a constant specifying the type of access with which to open the fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 908). You can open a path for writing even if it accesses a file on a locked volume, and no error is returned until a `PBWriteSync`, `PBSetEOFSync` (page 758) , or `PBAllocateSync` (page 568) call is made.

`ioDirID`

On input, the directory ID of the file's parent directory.

`ioFVersNum`

On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

Note that if you wish to access named forks other than the data and resource forks, or forks larger than 2GB, you will need to use the `FSOpenFork` (page 514) function, or one of its corresponding parameter block calls, `PBOpenForkSync` (page 740) or `PBOpenForkAsync` (page 739). If you try to open a fork larger than 2GB with the `PBHOpenDFSync` function, you will receive an error message.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Files.h`

PBHOpenRFAsync

Opens the resource fork of a file. (Deprecated in Mac OS X v10.4. Use `PBOpenForkAsync` (page 739) instead.)

```
OSErr PBHOpenRFAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the `HIOParam` (page 855) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function. On some file systems, `PBHOpenRFAsync` will return the error `eofErr` if you try to open the resource fork of a file for which no resource fork exists with read-only access.

`ioNamePtr`
On input, a pointer to the name of the file.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioRefNum`
On output, a file reference number for accessing the open resource fork.

`ioPermssn`
On input, a constant specifying the type of access with which to open the fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 908).

`ioDirID`
On input, the directory ID of the file's parent directory.

`ioFVersNum`
On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

Note that if you wish to access named forks other than the data and resource forks, or forks larger than 2GB, you will need to use the `FSOpenFork` (page 514) function, or one of its corresponding parameter block calls, `PBOpenForkSync` (page 740) or `PBOpenForkAsync` (page 739). If you try to open a fork larger than 2GB with the `PBHOpenRFAsync` function, you will receive an error message.

Special Considerations

Generally your application should use Resource Manager functions rather than File Manager functions to access a file's resource fork. The `PBHOpenRFAsync` function does not read the resource map into memory and is generally useful only for applications (such as utilities that copy files) that need block-level access to a resource fork.

You should not use the resource fork of a file to hold non-resource data. Many parts of the system software assume that a resource fork always contains resource data.

Because there is no support for locking and unlocking file ranges in Mac OS X, regardless of whether File Sharing is enabled, you cannot open more than one path to a resource fork with read/ write permission. If you try to open a more than one path to a file's resource fork with `fsRdWrShPerm` permission, only the first attempt will succeed. Subsequent attempts will return an invalid reference number and the `ResError` function will return the error `opWrErr`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBHOpenRFDenyAsync

Opens a file's resource fork using the access deny modes. (Deprecated in Mac OS X v10.5. Use `PBOpenForkAsync` (page 739) with deny modes in the permissions field.)

```
OSErr PBHOpenRFDenyAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `AccessParam` (page 795) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function. The function returns the result code `opWrErr` if you've requested write permission and you have already opened the file for writing in that case, the existing file reference number is returned in `ioRefNum`. You should not use this reference number unless your application originally opened the file.

`ioNamePtr`

On input, a pointer to a pathname for the file.

`ioVRefNum`
: On input, a volume reference number or drive number for the volume containing the file. Pass 0 to indicate the default volume.

`ioRefNum`
: On output, the file reference number for the file.

`ioDenyModes`
: On input, the type of access you are requesting to the fork. See "File Access Permission Constants" (page 908) for a description of the types of access that you can request.

`ioDirID`
: On input, the parent directory ID of the file.

You should use the `PBHOpenRFDenyAsync` and `PBHOpenDenyAsync` (page 704) functions (or their synchronous counterparts, `PBHOpenRFDenySync` (page 711) and `PBHOpenDenySync` (page 705)) if you want to ensure that you get the access permissions and deny-mode permissions that you request. `PBHOpenRFDenyAsync` is not retried in any way. If the file cannot be opened because of a deny conflict, the error `afpDenyConflict` is returned and the `ioRefNum` field is set to 0.

You can check that the volume supports AFP deny-mode permissions by checking that the `bHasOpenDeny` bit is set in the `vMAttrib` field returned by the `PBHGetVolParmsSync` (page 695) or `PBHGetVolParmsAsync` (page 694) function. If you don't want to special case volumes that support AFP deny mode permissions, you can use the File Manager permissions. See "File Access Permission Constants" (page 908) for a description of how File Manager permissions are translated to AFP deny-mode permissions.

To open a file's data fork with access deny permissions, use the `PBHOpenDenySync` (page 705) or `PBHOpenDenyAsync` (page 704) 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
`Files.h`

PBHOpenRFDenySync

Opens a file's resource fork using the access deny modes. (Deprecated in Mac OS X v10.5. Use `PBOpenForkSync` (page 740) with deny modes in the permissions field.)

```
OSErr PBHOpenRFDenySync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `AccessParam` (page 795) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943). The function returns the result code `opWrErr` if you've requested write permission and you have already opened the file for writing in that case, the existing file reference number is returned in `ioRefNum`. You should not use this reference number unless your application originally opened the file.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion function.

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to a pathname for the file.

`ioVRefNum`
On input, a volume reference number or drive number for the volume containing the file. Pass 0 to indicate the default volume.

`ioRefNum`
On output, the file reference number for the file.

`ioDenyModes`
On input, the type of access you are requesting to the fork. See "File Access Permission Constants" (page 908) for a description of the types of access that you can request.

`ioDirID`
On input, the parent directory ID of the file.

You should use the `PBHOpenRFDenySync` and `PBHOpenDenySync` (page 705) functions (or their asynchronous counterparts, `PBHOpenRFDenyAsync` (page 710) and `PBHOpenDenyAsync` (page 704)) if you want to ensure that you get the access permissions and deny-mode permissions that you request. `PBHOpenRFDenySync` is not retried in any way. If the file cannot be opened because of a deny conflict, the error `afpDenyConflict` is returned and the `ioRefNum` field is set to 0.

You can check that the volume supports AFP deny-mode permissions by checking that the `bHasOpenDeny` bit is set in the `vMAttrib` field returned by the `PBHGetVolParmsSync` (page 695) or `PBHGetVolParmsAsync` (page 694) function. If you don't want to special case volumes that support AFP deny mode permissions, you can use the File Manager permissions. See "File Access Permission Constants" (page 908) for a description of how File Manager permissions are translated to AFP deny-mode permissions.

To open a file's data fork with access deny permissions, use the `PBHOpenDenySync` (page 705) or `PBHOpenDenyAsync` (page 704) 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
`Files.h`

PBHOpenRFSync

Opens the resource fork of a file. (Deprecated in Mac OS X v10.4. Use `PBOpenForkSync` (page 740) instead.)

```
OSErr PBHOpenRFSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `HIOParam` (page 855) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943). On some file systems, `PBHOpenRFSync` will return the error `eofErr` if you try to open the resource fork of a file for which no resource fork exists with read-only access.

Discussion

The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to the name of the file.

`ioVRefNum`

On input, a volume reference number, drive number, or 0 for the default volume.

`ioRefNum`

On output, a file reference number for accessing the open resource fork.

`ioPermssn`

On input, a constant specifying the type of access with which to open the fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 908).

`ioDirID`

On input, the directory ID of the file's parent directory.

`ioFVersNum`

On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

Note that if you wish to access named forks other than the data and resource forks, or forks larger than 2GB, you will need to use the `FSOpenFork` (page 514) function, or one of its corresponding parameter block calls, `PBOpenForkSync` (page 740) or `PBOpenForkAsync` (page 739). If you try to open a fork larger than 2GB with the `PBOpenRFSync` function, you will receive an error message.

Special Considerations

Generally your application should use Resource Manager functions rather than File Manager functions to access a file's resource fork. The `PBHOpenRFSync` function does not read the resource map into memory and is generally useful only for applications (such as utilities that copy files) that need block-level access to a resource fork.

You should not use the resource fork of a file to hold non-resource data. Many parts of the system software assume that a resource fork always contains resource data.

Because there is no support for locking and unlocking file ranges on local disks in Mac OS X, regardless of whether File Sharing is enabled, you cannot open more than one path to a resource fork with read/ write permission. If you try to open a more than one path to a file's resource fork with `fsRdWrShPerm` permission, only the first attempt will succeed. Subsequent attempts will return an invalid reference number and the `ResError` function will return the error `opWrErr`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHOpenSync

Opens the data fork of a file. (Deprecated in Mac OS X v10.4. Use `PBOpenForkSync` (page 740) instead.)

```
OSErr PBHOpenSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to the `HIOParam` (page 855) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943). If you attempt to open a locked file for writing, `PBHOpenSync` returns the result code `permErr`. If you request exclusive read/write permission but another access path is already open, `PBHOpenSync` returns the reference number of the existing access path in `ioRefNum` and `opWrErr` as its function result.

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the name of the file.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioRefNum`
On output, a file reference number for accessing the open data fork. If you request exclusive read/write permission but another access path is already open, `PBHOpenSync` returns the reference number of the existing access path. You should not use this reference number unless your application originally opened the file.

`ioPermssn`
On input, a constant specifying the type of access with which to open the fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 908). You can open a path for writing even if it accesses a file on a locked volume, and no error is returned until a `PBWriteSync`, `PBSetEOFSync` (page 758) , or `PBAllocateSync` (page 568) call is made.

`ioDirID`
On input, the directory ID of the file's parent directory.

`ioFVersNum`
On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

If you use `PBHOpenSync` to try to open a file whose name begins with a period, you might mistakenly open a driver instead; subsequent attempts to write data might corrupt data on the target device. To avoid these problems, you should always use `PBHOpenDFSync` (page 708) instead of `PBHOpenSync`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHRenameAsync

Renames a file, directory, or volume. (Deprecated in Mac OS X v10.4. Use `PBRenameUnicodeAsync` (page 748) instead.)

```
OSErr PBHRenameAsync (
   HParmBlkPtr paramBlock
);
```

Parameters
`paramBlock`
A pointer to the `HIOParam` (page 855) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to the existing filename, directory name, or volume name.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioMisc`
On input, a pointer to the new name for the file, directory or volume.

`ioDirID`
On input, the parent directory ID of the file or directory to rename.

`ioFVersNum`
: On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

Given a pointer to the name of a file or directory in the `ioNamePtr` field, `PBHRenameAsync` changes it to the name pointed to in the `ioMisc` field. Given a pointer to a volume name in `ioNamePtr` or a volume reference number in `ioVRefNum`, the function changes the name of the volume to the name pointed to in `ioMisc`.

If a file ID reference exists for the file being renamed, the file ID remains with the file.

To rename a file or directory using a long Unicode name, use the `FSRenameUnicode` (page 539) function or one of the corresponding parameter block calls, `PBRenameUnicodeSync` (page 748) and `PBRenameUnicodeAsync` (page 748).

Special Considerations

You cannot use `PBHRenameAsync` to change the directory in which a file is located. To move a file or directory, use the `FSpCatMove` (page 524), `PBCatMoveSync` (page 576), or `PBCatMoveAsync` (page 575) 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
`Files.h`

PBHRenameSync

Renames a file, directory, or volume. (Deprecated in Mac OS X v10.4. Use `PBRenameUnicodeSync` (page 748) instead.)

```
OSErr PBHRenameSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `HIOParam` (page 855) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
: On input, a pointer to the existing filename, directory name, or volume name.

`ioVRefNum`
: On input, a volume reference number, drive number, or 0 for the default volume.

`ioMisc`
: On input, a pointer to the new name for the file, directory or volume.

`ioDirID`
On input, the parent directory ID of the file or directory to rename.

`ioFVersNum`
On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

Given a pointer to the name of a file or directory in the `ioNamePtr` field, `PBHRenameSync` changes it to the name pointed to in the `ioMisc` field. Given a pointer to a volume name in `ioNamePtr` or a volume reference number in `ioVRefNum`, the function changes the name of the volume to the name pointed to in `ioMisc`.

If a file ID reference exists for the file being renamed, the file ID remains with the file.

To rename a file or directory using a long Unicode name, use the `FSRenameUnicode` (page 539) function or one of the corresponding parameter block calls, `PBRenameUnicodeSync` (page 748) and `PBRenameUnicodeAsync` (page 748).

Special Considerations

You cannot use `PBHRenameSync` to change the directory in which a file is located. To move a file or directory, use the `FSpCatMove` (page 524), `PBCatMoveSync` (page 576), or `PBCatMoveAsync` (page 575) 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
`Files.h`

PBHRstFLockAsync

Unlocks a file or directory. (Deprecated in Mac OS X v10.4. Use `PBSetCatalogInfoAsync` (page 751) instead.)

```
OSErr PBHRstFLockAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to the `HFileParam` (page 852) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to the name for the file or directory o unlock.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioDirID`
On input, the parent directory ID of the file or directory to unlock.

`ioFVersNum`
On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

If the `PBHGetVolParmsSync` (page 695) or `PBHGetVolParmsAsync` (page 694) function indicates that the volume supports folder locking (that is, the `bHasFolderLock` bit of the `vMAttrib` field is set), you can use `PBHRstFLockAsync` to unlock a directory. Otherwise, you can only use this function to unlock a file.

Access paths currently in use aren't affected by 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
`Files.h`

PBHRstFLockSync

Unlocks a file or directory. (Deprecated in Mac OS X v10.4. Use `PBSetCatalogInfoSync` (page 753) instead.)

```
OSErr PBHRstFLockSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to the `HFileParam` (page 852) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the name for the file or directory to unlock.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioDirID`
On input, the parent directory ID of the file or directory to unlock.

`ioFVersNum`
On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

If the PBHGetVolParmsSync (page 695) or PBHGetVolParmsAsync (page 694) function indicates that the volume supports folder locking (that is, the bHasFolderLock bit of the vMAttrib field is set), you can use PBHRstFLockSync to unlock a directory. Otherwise, you can only use this function to unlock a file.

Access paths currently in use aren't affected by 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
Files.h

PBHSetDirAccessAsync

Changes the access control information for a directory. (Deprecated in Mac OS X v10.5. Use FSSetCatalogInfo (page 540) instead.)

```
OSErr PBHSetDirAccessAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to an AccessParam (page 795) variant of an HFS parameter block. See HParamBlockRec (page 857) for a description of the HParamBlockRec data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

ioCompletion
On input, a pointer to a completion function. For more information on completion routines, see IOCompletionProcPtr (page 794).

ioResult
On output, the result code of the function.

ioNamePtr
On input, a pointer to a pathname.

ioVRefNum
On input, a volume specification for the volume containing the directory. This field can contain a volume reference number, drive number, or 0 for the default volume.

ioACOwnerID
On input, the owner ID.

ioACGroupID
On input, the group ID.

ioACAccess
On input, the directory's access rights. You cannot set the owner or user rights bits of the ioACAccess field directly; if you try to do this, PBHSetDirAccessAsync returns the result code paramErr. Only

the blank access privileges can be set for a directory using this function. See "File and Folder Access Privilege Constants" (page 910) for more information on directory access privileges.

`ioDirID`
On input, the directory ID.

To change the owner or group, you should set the `ioACOwnerID` or `ioACGroupID` field to the appropriate ID. You must be the owner of the directory to change the owner or group ID. A guest on a server can manipulate the privileges of any directory owned by the guest.

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
`Files.h`

PBHSetDirAccessSync

Changes the access control information for a directory. (Deprecated in Mac OS X v10.5. Use `FSSetCatalogInfo` (page 540) instead.)

```
OSErr PBHSetDirAccessSync (
   HParmBlkPtr paramBlock
);
```

Parameters
`paramBlock`
A pointer to an `AccessParam` (page 795) variant of an HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a pathname.

`ioVRefNum`
On input, a volume specification for the volume containing the directory. This field can contain a volume reference number, drive number, or 0 for the default volume.

`ioACOwnerID`
On input, the owner ID.

`ioACGroupID`
On input, the group ID.

`ioACAccess`
On input, the directory's access rights. You cannot set the owner or user rights bits of the `ioACAccess` field directly; if you try to do this, `PBHSetDirAccessSync` returns the result code `paramErr`. Only the blank access privileges can be set for a directory using this function. See "File and Folder Access Privilege Constants" (page 910) for more information on directory access privileges.

`ioDirID`
On input, the directory ID.

To change the owner or group, you should set the `ioACOwnerID` or `ioACGroupID` field to the appropriate ID. You must be the owner of the directory to change the owner or group ID. A guest on a server can manipulate the privileges of any directory owned by the guest.

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
`Files.h`

PBHSetFInfoAsync

Sets information for a file. (Deprecated in Mac OS X v10.4. Use `PBSetCatalogInfoAsync` (page 751) instead.)

```
OSErr PBHSetFInfoAsync (
   HParmBlkPtr paramBlock
);
```

Parameters
`paramBlock`
A pointer to the `HFileParam` (page 852) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to the name of the file.

`ioVRefNum`
On input, the volume reference number or drive number for the volume containing the file; or 0 for the default volume.

`ioFlFndrInfo`
On input, Finder information for the file. For a description of the `FInfo` data type, see the *Finder Interface Reference* .

`ioDirID`
On input, the parent directory ID for the file.

`ioFVersNum`
: On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

`ioFlCrDat`
: On input, the date and time of the file's creation.

`ioFlMdDat`
: On input, the date and time of the file's last modification.

You should call the `PBHGetFInfoAsync` (page 682) function just before calling `PBHSetFInfoAsync`, so that the current information is present in the parameter block.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHSetFInfoSync

Sets information for a file. (Deprecated in Mac OS X v10.4. Use `PBSetCatalogInfoSync` (page 753) instead.)

```
OSErr PBHSetFInfoSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `HFileParam` (page 852) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
: On input, a pointer to the name of the file.

`ioVRefNum`
: On input, the volume reference number or drive number for the volume containing the file; or 0 for the default volume.

`ioFlFndrInfo`
: On input, Finder information for the file. For a description of the `FInfo` data type, see the *Finder Interface Reference* .

`ioDirID`
: On input, the parent directory ID of the file.

`ioFVersNum`
: On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

`ioFlCrDat`
On input, the date and time of the file's creation.

`ioFlMdDat`
On input, the date and time of the file's last modification.

You should call the `PBHGetFInfoSync` (page 683) function just before calling `PBHSetFInfoSync`, so that the current information is present in the parameter block.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHSetFLockAsync

Locks a file or directory. (Deprecated in Mac OS X v10.4. Use `PBSetCatalogInfoAsync` (page 751) instead.)

```
OSErr PBHSetFLockAsync (
   HParmBlkPtr paramBlock
);
```

Parameters
`paramBlock`
A pointer to the `HFileParam` (page 852) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to a name for the file or directory to lock.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioDirID`
On input, the parent directory ID of the file or directory to lock.

`ioFVersNum`
On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

If the PBHGetVolParmsSync (page 695) or PBHGetVolParmsAsync (page 694) function indicates that the volume supports folder locking (that is, the bHasFolderLock bit of the vMAttrib field is set), you can use PBHSetFLockAsync to lock a directory. Otherwise, you can only use this function to lock a file.

After you lock a file, all new access paths to that file are read-only. Access paths currently in use aren't affected.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Files.h

PBHSetFLockSync

Locks a file or directory. (Deprecated in Mac OS X v10.4. Use PBSetCatalogInfoSync (page 753) instead.)

```
OSErr PBHSetFLockSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the HFileParam (page 852) variant of the basic HFS parameter block. See HParamBlockRec (page 857) for a description of the HParamBlockRec data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

ioNamePtr
On input, a pointer to a name for the file or directory to lock.

ioVRefNum
On input, a volume reference number, drive number, or 0 for the default volume.

ioDirID
On input, the parent directory ID of the file or directory to lock.

ioFVersNum
On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

If the PBHGetVolParmsSync (page 695) or PBHGetVolParmsAsync (page 694) function indicates that the volume supports folder locking (that is, the bHasFolderLock bit of the vMAttrib field is set), you can use PBHSetFLockSync to lock a directory. Otherwise, you can only use this function to lock a file.

After you lock a file, all new access paths to that file are read-only. Access paths currently in use aren't affected.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHSetVolAsync

Sets the default volume and the default directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBHSetVolAsync (
   WDPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a working directory parameter block. See `WDPBRec` (page 877) for a description of the `WDPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`ioNamePtr`

On input, a pointer to a pathname. If this field specifies a full pathname, the default volume is set to the volume whose name is contained in that pathname. (A full pathname overrides the `ioVRefNum` field.)If this field contains a partial pathname and the `ioVRefNum` field specifies a volume reference number, then the default directory is set to the directory having the partial pathname specified here, in the directory given in the `ioWDDirID` field. If this field is `NULL`, then the default directory is set to the directory having the ID specified in the `ioWDDirID` field.

`ioVRefNum`

On input, a volume reference number for the default volume. This field is ignored if the `ioNamePtr` field specifies a full pathname.

`ioWDDirID`

On input, a directory ID. If the `ioVRefNum` field contains a volume reference number and `ioNamePtr` contains a partial pathname, this field contains the directory ID of the directory containing the default directory. If `ioNamePtr` is `NULL`, this field contains the directory ID of the default directory.

Both the default volume and the default directory are used in calls made with no volume name, a volume reference number of 0, and a directory ID of 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
`Files.h`

PBHSetVolSync

Sets the default volume and the default directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBHSetVolSync (
   WDPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a working directory parameter block. See `WDPBRec` (page 877) for a description of the `WDPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to a pathname. If this field specifies a full pathname, the default volume is set to the volume whose name is contained in that pathname. (A full pathname overrides the `ioVRefNum` field.)If this field contains a partial pathname and the `ioVRefNum` field specifies a volume reference number, then the default directory is set to the directory having the partial pathname specified here, in the directory given in the `ioWDDirID` field. If this field is `NULL`, then the default directory is set to the directory having the ID specified in the `ioWDDirID` field.

`ioVRefNum`

On input, the volume reference number for the default volume. This field is ignored if the `ioNamePtr` field specifies a full pathname.

`ioWDDirID`

On input, a directory ID. If the `ioVRefNum` field contains a volume reference number and `ioNamePtr` contains a partial pathname, this field contains the directory ID of the directory containing the default directory. If `ioNamePtr` is `NULL`, this field contains the directory ID of the default directory.

Both the default volume and the default directory are used in calls made with no volume name, a volume reference number of 0, and a directory ID of 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
`Files.h`

PBIterateForksAsync

Determines the name and size of every named fork belonging to a file or directory.

```
void PBIterateForksAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for more information on the `FSForkIOParam` data type.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`ref`

On input, a pointer to an `FSRef` (page 837) specifying the file or directory to iterate.

`forkIterator`

A pointer to a structure which maintains state between calls to `PBIterateForksAsync`. Before the first call, set the `initialize` field of this structure to 0. The fork iterator will be updated after the call completes; the updated iterator should be passed into the next call. See `CatPositionRec` (page 801) for a description of the structure pointed to in this field.

`outForkName`

On output, a pointer to the Unicode name of the fork.

`positionOffset`

On output, the logical size of the fork, in bytes.

`allocationAmount`

On output, the fork's physical size (that is, the amount of space allocated on disk), in bytes.

Since information is returned about one fork at a time, several calls may be required to iterate through all the forks. There is no guarantee about the order in which forks are returned; the order may vary between iterations.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBIterateForksSync

Determines the name and size of every named fork belonging to a file or directory.

```
OSErr PBIterateForksSync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for more information on the `FSForkIOParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ref`

On input, a pointer to an `FSRef` (page 837) specifying the file or directory to iterate.

`forkIterator`

A pointer to a structure which maintains state between calls to `PBIterateForksSync`. Before the first call, set the `initialize` field of this structure to 0. The fork iterator will be updated after the call completes; the updated iterator should be passed into the next call. See `CatPositionRec` (page 801) for a description of the structure pointed to in this field.

`outForkName`

On output, a pointer to the Unicode name of the fork.

`positionOffset`

On output, the logical size of the fork, in bytes.

`allocationAmount`

On output, the fork's physical size (that is, the amount of space allocated on disk), in bytes.

Since information is returned about one fork at a time, several calls may be required to iterate through all the forks. There is no guarantee about the order in which forks are returned; the order may vary between iterations.

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

Declared In
`Files.h`

PBLockRangeAsync

Locks a portion of a file. (Deprecated in Mac OS X v10.4. Use `PBXLockRangeAsync` (page 785) instead.)

```
OSErr PBLockRangeAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `IOParam` (page 862) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 866) for a description of the `ParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function. If you call `PBLockRangeAsync` on a file system that does not implement it—for example, SMB—`PBLockRangeAsync` returns `noErr` and does nothing.

`ioRefNum`
: On input, the file reference number of the file owning the range to lock.

`ioReqCount`
: On input, the number of bytes in the range. Set `ioReqCount` to –1 to lock the maximum number of bytes from the position specified in the `ioPosOffset` field.

`ioPosMode`
: On input, a constant specifying the base location for the start of the locked range. See "Position Mode Constants" (page 928) for more information on the constants you can use to specify the base location.

  You should not use the `fsFromLEOF` constant when locking a file range. `PBLockRangeAsync` does not return the start of the locked range; thus, there is no way to determine what range was actually locked when you specify `fsFromLEOF`.

`ioPosOffset`
: On input, the offset from the base location specified in the `ioPosMode` field for the start of the locked range.

The `PBLockRangeAsync` function locks a portion of a file that was opened with shared read/write permission. The beginning of the range to be locked is determined by the `ioPosMode` and `ioPosOffset` fields. The end of the range to be locked is determined by the beginning of the range and the `ioReqCount` field. For example, to lock the first 50 bytes in a file, set `ioReqCount` to 50, `ioPosMode` to `fsFromStart`, and `ioPosOffset` to 0.

The `PBLockRangeAsync` function uses the same parameters as both `PBReadAsync` and `PBWriteAsync`; by calling it immediately before `PBReadAsync`, you can use the information in the parameter block for the `PBReadAsync` call.

When you're finished with the data (typically after a call to `PBWriteSync`), you can call `PBUnlockRangeAsync` (page 770) to free that portion of the file for subsequent read and write calls. Closing a file also releases all locked ranges in that file.

Special Considerations

The `PBLockRangeAsync` function does nothing if the file specified in the `ioRefNum` field is open with shared read/write permission but is not located on a remote server volume or is not located under a share point on a sharable local volume. To check whether file sharing is currently on, check that the `bHasPersonalAccessPrivileges` bit in the `vMAttrib` field of the `GetVolParmsInfoBuffer` (page 847) returned by the `PBHGetVolParmsSync` (page 695) function is set.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
Files.h

PBLockRangeSync

Locks a portion of a file. (Deprecated in Mac OS X v10.4. Use PBXLockRangeSync (page 785) or FSLockRange (page 504) instead.)

```
OSErr PBLockRangeSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the IOParam (page 862) variant of the basic File Manager parameter block. See ParamBlockRec (page 866) for a description of the ParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 943). If you call PBLockRangeSync on a file system that does not implement it—for example, SMB—PBLockRangeSync returns noErr and does nothing.

Discussion

The relevant fields of the parameter block are:

ioRefNum

On input, the file reference number of the file owning the range to lock.

ioReqCount

On input, the number of bytes in the range. Set ioReqCount to –1 to lock the maximum number of bytes from the position specified in the ioPosOffset field.

ioPosMode

On input, a constant specifying the base location for the start of the locked range. See "Position Mode Constants" (page 928) for more information about the constants you can use to specify the base location.

You should not use the fsFromLEOF constant when locking a file range. PBLockRangeSync does not return the start of the locked range; thus, there is no way to determine what range was actually locked when you specify fsFromLEOF.

ioPosOffset

On input, the offset from the base location specified in the ioPosMode field for the start of the locked range.

The PBLockRangeSync function locks a portion of a file that was opened with shared read/write permission. The beginning of the range to be locked is determined by the ioPosMode and ioPosOffset fields. The end of the range to be locked is determined by the beginning of the range and the ioReqCount field. For example, to lock the first 50 bytes in a file, set ioReqCount to 50, ioPosMode to fsFromStart, and ioPosOffset to 0.

The PBLockRangeSync function uses the same parameters as both PBReadSync and PBWriteSync; by calling it immediately before PBReadSync, you can use the information in the parameter block for the PBReadSync call.

When you're finished with the data (typically after a call to PBWriteSync), you can call PBUnlockRangeSync (page 771) to free that portion of the file for subsequent read and write calls. Closing a file also releases all locked ranges in that file.

Special Considerations

The `PBLockRangeSync` function does nothing if the file specified in the `ioRefNum` field is open with shared read/write permission but is not located on a remote server volume or is not located under a share point on a sharable local volume. To check whether file sharing is currently on, check that the `bHasPersonalAccessPrivileges` bit in the `vMAttrib` field of the `GetVolParmsInfoBuffer` (page 847) returned by the `PBHGetVolParmsSync` (page 695) function is set.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBMakeFSRefAsync

Creates an `FSRef` for a file or directory, given an `FSSpec`. (Deprecated in Mac OS X v10.5. Use `PBMakeFSRefUnicodeAsync` (page 733) instead.)

```
void PBMakeFSRefAsync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 837) for a description of the `FSRefParam` data type.

Discussion

For the parameter block based calls, the fields of the source `FSSpec` are passed as separate parameters (in the `ioNamePtr`, `ioVRefNum`, and `ioDirID` fields). This allows the call to be dispatched to external file systems the same way as other `FSp` calls are.

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`ioNamePtr`

On input, a pointer to the name of the file or directory for which you wish to create an `FSRef`.

`ioVRefNum`

On input, a volume specification for the volume containing the file or directory. This can be a volume reference number, a drive number, or 0 for the default volume.

`ioDirID`

On input, the directory ID of the file or directory's parent directory.

`newRef`

On input, a pointer to an `FSRef` structure. On output, this `FSRef` refers to the specified file or directory.

To obtain an FSSpec from an FSRef, use the PBGetCatalogInfoAsync (page 643) call.

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
Files.h

PBMakeFSRefSync

Creates an FSRef for a file or directory, given an FSSpec. (Deprecated in Mac OS X v10.5. Use PBMakeFSRefUnicodeSync (page 733) instead.)

```
OSErr PBMakeFSRefSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock
A pointer to a file system reference parameter block. See FSRefParam (page 837) for a description of the FSRefParam data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
For the parameter block based calls, the fields of the source FSSpec are passed as separate parameters (in the ioNamePtr, ioVRefNum, and ioDirID fields). This allows the call to be dispatched to external file systems the same way as other FSp calls are.

The relevant fields of the parameter block are:

ioNamePtr
On input, a pointer to the name of the file or directory for which you wish to create an FSRef.

ioVRefNum
On input, a volume specification for the volume containing the file or directory. This can be a volume reference number, a drive number, or 0 for the default volume.

ioDirID
On input, the directory ID of the file or directory's parent directory.

newRef
On input, a pointer to an FSRef structure. On output, this FSRef refers to the specified file or directory.

To obtain an FSSpec from an FSRef, use the PBGetCatalogInfoSync (page 647) 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
Files.h

PBMakeFSRefUnicodeAsync

Constructs an FSRef for a file or directory, given a parent directory and a Unicode name.

```
void PBMakeFSRefUnicodeAsync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock
A pointer to a file system reference parameter block. See FSRefParam (page 837) for a description of the FSRefParam data type.

Discussion
The relevant fields of the parameter block are:

ioCompletion
On input, a pointer to a completion function. For more information on completion routines, see IOCompletionProcPtr (page 794).

ioResult
On output, the result code of the function.

ref
On input, a pointer to the FSRef of the parent directory of the file or directory for which to create a new FSRef. See FSRef (page 837) for a description of the FSRef data type.

nameLength
On input, the length of the file or directory name.

name
On input, a pointer to the Unicode name for the file or directory. The name must be a leaf name; partial or full pathnames are not allowed. If you have a partial or full pathname in Unicode, you will have to parse it yourself and make multiple calls to PBMakeFSRefUnicodeAsync.

textEncodingHint
On input, the suggested text encoding to use when converting the Unicode name of the file or directory to some other encoding. If you pass the constant kTextEncodingUnknown, the File Manager will use a default value.

newRef
On output, if the function returns a result of noErr, a pointer to the new FSRef

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

Declared In
Files.h

PBMakeFSRefUnicodeSync

Constructs an FSRef for a file or directory, given a parent directory and a Unicode name.

```
OSErr PBMakeFSRefUnicodeSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 837) for a description of the `FSRefParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ref`

On input, a pointer to the `FSRef` of the parent directory of the file or directory for which to create a new `FSRef`. See `FSRef` (page 837) for a description of the `FSRef` data type.

`nameLength`

On input, the length of the file or directory name.

`name`

On input, a pointer to the Unicode name for the file or directory. The name must be a leaf name; partial or full pathnames are not allowed. If you have a partial or full pathname in Unicode, you will have to parse it yourself and make multiple calls to `PBMakeFSRefUnicodeSync`.

`textEncodingHint`

On input, the suggested text encoding to use when converting the Unicode name of the file or directory to some other encoding. If you pass the constant `kTextEncodingUnknown`, the File Manager will use a default value.

`newRef`

On output, if the function returns a result of `noErr`, a pointer to the new `FSRef`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBMakeFSSpecAsync

Creates an `FSSpec` structure for a file or directory. (Deprecated in Mac OS X v10.4. Use `PBMakeFSRefUnicodeAsync` (page 733) instead.)

```
OSErr PBMakeFSSpecAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

If the specified volume is mounted and the specified parent directory exists, but the target file or directory doesn't exist in that location, PBMakeFSSpecAsync fills in the structure and returns fnfErr instead of noErr. The structure is valid, but it describes a target that doesn't exist. You can use the structure for another operation, such as creating a file.

PBMakeFSSpecAsync can return a number of different File Manager error codes. When PBMakeFSSpecAsync returns any result other than noErr or fnfErr, all fields of the resulting FSSpec structure are set to 0.

Discussion
The relevant fields of the parameter block are:

ioCompletion
: On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 794).

ioResult
: On output, the result code of the function. When PBMakeFSSpecAsync returns any result other than noErr or fnfErr, all fields of the resulting FSSpec structure are set to 0. See "File Manager Result Codes".

ioNamePtr
: On input, a pointer to a full or partial pathname specifying the file or directory for which to create an FSSpec. If the ioNamePtr field specifies a full pathname, PBMakeFSSpecAsync ignores both the ioVRefNum and ioDirID fields. A partial pathname might identify only the final target, or it might include one or more parent directory names. If ioNamePtr specifies a partial pathname, then ioVRefNum, ioDirID, or both must be valid.

ioVRefNum
: On input, a volume specification for the volume containing the file or directory. This field can contain a volume reference number, a drive number, or 0 to specify the default volume.

ioMisc
: On input, a pointer to an FSSpec (page 840) structure. Given a complete specification for a file or directory, the PBMakeFSSpecAsync function fills in this FSSpec structure to identify the file or directory. On output, this field points to the initialized FSSpec. The file system specification structure that you pass in this field should not share storage space with the input pathname; the name field may be initialized to the empty string before the pathname has been processed. For example, ioNamePtr should not refer to the name field of the output file system specification.

ioDirID
: On input, a directory ID. This field usually specifies the parent directory ID of the target object. If the directory is sufficiently specified by the ioNamePtr field, the ioDirID field can be set to 0. If the ioNamePtr field contains an empty string, PBMakeFSSpecAsync creates an FSSpec structure for the directory specified by the ioDirID field.

If the specified volume is mounted and the specified parent directory exists, but the target file or directory doesn't exist in that location, PBMakeFSSpecAsync fills in the structure and returns fnfErr instead of noErr. The structure is valid, but it describes a target that doesn't exist. You can use the structure for another operation, such as creating a file.

Carbon Porting Notes

Non-Carbon applications can also specify a working directory reference number in the ioVRefNum field. However, because working directories are not supported in Carbon, you cannot specify a working directory reference number if you wish your application to be Carbon-compatible.

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

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBMakeFSSpecSync

Creates an `FSSpec` structure for a file or directory. (Deprecated in Mac OS X v10.4. Use `PBMakeFSRefUnicodeSync` (page 733) instead.)

```
OSErr PBMakeFSSpecSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to a basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943). When `PBMakeFSSpecSync` returns any result other than `noErr` or `fnfErr`, all fields of the resulting `FSSpec` structure are set to 0.

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to a full or partial pathname specifying the file or directory for which to create an `FSSpec`. If the `ioNamePtr` field specifies a full pathname, `PBMakeFSSpecSync` ignores both the `ioVRefNum` and `ioDirID` fields. A partial pathname might identify only the final target, or it might include one or more parent directory names. If `ioNamePtr` specifies a partial pathname, then `ioVRefNum`, `ioDirID`, or both must be valid.

`ioVRefNum`

On input, a volume specification for the volume containing the file or directory. This field can contain a volume reference number, a drive number, or 0 to specify the default volume.

`ioMisc`

On input, a pointer to an `FSSpec` (page 840) structure. Given a complete specification for a file or directory, the `PBMakeFSSpecSync` function fills in this `FSSpec` structure to identify the file or directory. On output, this field points to the initialized `FSSpec`. The file system specification structure that you pass in this field should not share storage space with the input pathname; the `name` field may be initialized to the empty string before the pathname has been processed. For example, `ioNamePtr` should not refer to the `name` field of the output file system specification.

`ioDirID`

On input, a directory ID. This field usually specifies the parent directory ID of the target object. If the directory is sufficiently specified by the `ioNamePtr` field, the `ioDirID` field can be set to 0. If the `ioNamePtr` field contains an empty string, `PBMakeFSSpecSync` creates an `FSSpec` structure for the directory specified by the `ioDirID` field.

If the specified volume is mounted and the specified parent directory exists, but the target file or directory doesn't exist in that location, `PBMakeFSSpecSync` fills in the structure and returns `fnfErr` instead of `noErr`. The structure is valid, but it describes a target that doesn't exist. You can use the structure for another operation, such as creating a file.

Carbon Porting Notes

Non-Carbon applications can also specify a working directory reference number in the `ioVRefNum` field. However, because working directories are not supported in Carbon, you cannot specify a working directory reference number if you wish your application to be Carbon-compatible.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBMoveObjectAsync

Moves a file or directory into a different directory.

```
void PBMoveObjectAsync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 837) for a description of the `FSRefParam` data type.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function. If the `parentRef` field specifies a non-existent object, `dirNFErr` is returned; if it refers to a file, then `errFSNotAFolder` is returned. If the directory specified in `parentRef` is on a different volume than the file or directory indicated by the `ref` field, `diffVolErr` is returned.

`ref`

On input, a pointer to an `FSRef` (page 837) specifying the file or directory to move.

parentRef

On input, a pointer to an `FSRef` specifying the directory into which the file or directory given in the `ref` field will be moved.

`newRef`

On output, a pointer to the new `FSRef` for the file or directory in its new location. This field is optional; if you do not wish the `FSRef` returned, pass `NULL` here.

Moving an object may change its FSRef. If you want to continue to refer to the object, you should pass a non- NULL pointer in the newRef field and use the FSRef returned there to refer to the object after the move. The original FSRef passed in the ref field may or may not be usable after the move. The newRef field may point to the same storage as the parentRef or ref fields.

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

Declared In
Files.h

PBMoveObjectSync

Moves a file or directory into a different directory.

```
OSErr PBMoveObjectSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock
A pointer to a file system reference parameter block. See FSRefParam (page 837) for a description of the FSRefParam data type.

Return Value
A result code. See "File Manager Result Codes" (page 943). If the the parentRef field of the parameter block specifies a non-existent object, dirNFErr is returned; if it refers to a file, errFSNotAFolder is returned. If the directory specified in the parentRef field is on a different volume than the file or directory indicated in the ref field, diffVolErr is returned.

Discussion
The relevant fields of the parameter block are:

ioResult
On output, the result code of the function. If the parentRef field specifies a non-existent object, dirNFErr is returned; if it refers to a file, then errFSNotAFolder is returned. If the directory specified in parentRef is on a different volume than the file or directory indicated by the ref field, diffVolErr is returned.

ref
On input, a pointer to an FSRef (page 837) specifying the file or directory to move.

parentRef
On input, a pointer to an FSRef specifying the directory into which the file or directory given in the ref field will be moved.

newRef
On output, a pointer to the new FSRef for the file or directory in its new location. This field is optional; if you do not wish the FSRef returned, pass NULL here.

Moving an object may change its FSRef. If you want to continue to refer to the object, you should pass a non- NULL pointer in the newRef field and use the FSRef returned there to refer to the object after the move. The original FSRef passed in the ref field may or may not be usable after the move. The newRef field may point to the same storage as the parentRef or ref fields.

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

Declared In
`Files.h`

PBOpenForkAsync

Opens any fork of a file or directory for streaming access.

```
void PBOpenForkAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function. On some file systems, `PBOpenForkAsync` will return the error `eofErr` if you try to open the resource fork of a file for which no resource fork exists with read-only access.

`ref`

On input, a pointer to an `FSRef` (page 837) specifying the file or directory that owns the fork to open.

`forkNameLength`

On input, the length of the fork's Unicode name.

`forkName`

On input, a pointer to the Unicode name of the fork to open. You can obtain the string constants for the data and resource fork names using the `FSGetDataForkName` (page 497) and `FSGetResourceForkName` (page 500) functions. All volume formats should support data and resource forks; other named forks may be supported by some volume formats.

`permissions`

On input, a constant indicating the type of access that you wish to have to the fork via the returned fork reference. This parameter is the same as the `permission` parameter passed to the `FSpOpenDF` and `FSpOpenRF` functions. For a description of the types of access which you can request, see "File Access Permission Constants" (page 908).

`forkRefNum`

On output, the fork reference number for accessing the open fork.

If you wish to access named forks or forks larger than 2GB you must use the `FSOpenFork` function or one of the corresponding parameter block calls, `PBOpenForkSync` and `PBOpenForkAsync`. To determine if the `PBOpenForkSync` function is present, call the `Gestalt` function.

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

Declared In
Files.h

PBOpenForkSync

Opens any fork of a file or directory for streaming access.

```
OSErr PBOpenForkSync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock
A pointer to a fork I/O parameter block. See FSForkIOParam (page 833) for a description of the FSForkIOParam data type.

Return Value

A result code. See "File Manager Result Codes" (page 943). On some file systems, PBOpenForkSync will return the error eofErr if you try to open the resource fork of a file for which no resource fork exists with read-only access.

Discussion

The relevant fields of the parameter block are:

ref
On input, a pointer to an FSRef (page 837) specifying the file or directory that owns the fork to open.

forkNameLength
On input, the length of the fork's Unicode name.

forkName
On input, a pointer to the Unicode name of the fork to open. You can obtain the string constants for the data and resource fork names using the FSGetDataForkName (page 497) and FSGetResourceForkName (page 500) functions. All volume formats should support data and resource forks; other named forks may be supported by some volume formats.

permissions
On input, a constant indicating the type of access that you wish to have to the fork via the returned fork reference. This parameter is the same as the permission parameter passed to the FSpOpenDF and FSpOpenRF functions. For a description of the types of access which you can request, see "File Access Permission Constants" (page 908).

forkRefNum
On output, the fork reference number for accessing the open fork.

If you wish to access named forks or forks larger than 2GB you must use the FSOpenFork function or one of the corresponding parameter block calls, PBOpenForkSync and PBOpenForkAsync. To determine if the PBOpenForkSync function is present, call the Gestalt function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Files.h

PBOpenIteratorAsync

Creates a catalog iterator that can be used to iterate over the contents of a directory or volume.

```
void PBOpenIteratorAsync (
   FSCatalogBulkParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a catalog information parameter block. See `FSCatalogBulkParam` (page 824) for a description of the `FSCatalogBulkParam` data type.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`iterator`

On output, the new `FSIterator` (page 835). You can pass this iterator to the `FSGetCatalogInfoBulk` (page 495) or `FSCatalogSearch` (page 472) functions and their parameter block-based counterparts. The iterator is automatically initialized so that the next use of the iterator returns the first item. The order that items are returned in is volume format dependent and may be different for two different iterators created with the same container and flags.

`iteratorFlags`

On input, a set of flags which controls whether the iterator iterates over subtrees or just the immediate children of the container. See "Iterator Flags" (page 924) for a description of the flags defined for this field. Iteration over subtrees which do not originate at the root directory of a volume are not currently supported, and passing the `kFSIterateSubtree` flag in this field returns `errFSBadIteratorFlags`. To determine whether subtree iterators are supported, check that the `bSupportsSubtreeIterators` bit returned by `PBHGetVolParmsAsync` (page 694) is set.

`container`

On input, a pointer to an `FSRef` (page 837) for the directory to iterate. The set of items to iterate over can either be the objects directly contained in the directory, or all items directly or indirectly contained in the directory (in which case, the specified directory is the root of the subtree to iterate).

Catalog iterators must be closed when you are done using them, whether or not you have iterated over all the items. Iterators are automatically closed upon process termination, just like open files. However, you should use the `FSCloseIterator` (page 475) function, or one of the related parameter block functions, `PBCloseIteratorSync` (page 584) and `PBCloseIteratorAsync` (page 584) , to close an iterator to free up any system resources allocated to the iterator.

Before calling this function, you should check that it is present, by calling the `Gestalt` function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBOpenIteratorSync

Creates a catalog iterator that can be used to iterate over the contents of a directory or volume.

```
OSErr PBOpenIteratorSync (
   FSCatalogBulkParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a catalog information parameter block. See `FSCatalogBulkParam` (page 824) for a description of the `FSCatalogBulkParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`iterator`

On output, the new `FSIterator` (page 835). You can pass this iterator to the `FSGetCatalogInfoBulk` (page 495) or `FSCatalogSearch` (page 472) functions and their parameter block-based counterparts. The iterator is automatically initialized so that the next use of the iterator returns the first item. The order that items are returned in is volume format dependent and may be different for two different iterators created with the same container and flags.

`iteratorFlags`

On input, a set of flags which controls whether the iterator iterates over subtrees or just the immediate children of the container. See "Iterator Flags" (page 924) for a description of the flags defined for this field. Iteration over subtrees which do not originate at the root directory of a volume are not currently supported, and passing the `kFSIterateSubtree` flag in this field returns `errFSBadIteratorFlags`. To determine whether subtree iterators are supported, check that the `bSupportsSubtreeIterators` bit returned by `PBHGetVolParmsSync` (page 695) is set.

`container`

On input, a pointer to an `FSRef` (page 837) for the directory to iterate. The set of items to iterate over can either be the objects directly contained in the directory, or all items directly or indirectly contained in the directory (in which case, the specified directory is the root of the subtree to iterate).

Catalog iterators must be closed when you are done using them, whether or not you have iterated over all the items. Iterators are automatically closed upon process termination, just like open files. However, you should use the `FSCloseIterator` (page 475) function, or one of the related parameter block functions, `PBCloseIteratorSync` (page 584) and `PBCloseIteratorAsync` (page 584) , to close an iterator to free up any system resources allocated to the iterator.

Before calling this function, you should check that it is present, by calling the `Gestalt` function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBReadAsync

Reads any number of bytes from an open file. (Deprecated in Mac OS X v10.5. Use `PBReadForkAsync` (page 744) instead.)

```
OSErr PBReadAsync (
    ParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to a basic File Manager parameter block.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine.

`ioResult`
On output, the result code of the function.

`ioRefNum`
On input, a file reference number for an open file to be read.

`ioBuffer`
On input, a pointer to a data buffer into which the bytes are read.

`ioReqCount`
On input, the number of bytes requested. The value that you pass in this field should be greater than zero.

`ioActCount`
On output, the number of bytes actually read.

`ioPosMode`
On input, the positioning mode.

`ioPosOffset`
On input, the positioning offset. On output, the new position of the mark.

This function attempts to read `ioReqCount` bytes from the open file whose access path is specified in the `ioRefNum` field and transfer them to the data buffer pointed to by the `ioBuffer` field. The position of the mark is specified by `ioPosMode` and `ioPosOffset`. If your application tries to read past the logical end-of-file, `PBReadAsync` reads the data, moves the mark to the end-of-file, and returns `eofErr` as its function result. Otherwise, `PBReadAsync` moves the file mark to the byte following the last byte read and returns `noErr`.

You can specify that `PBReadAsync` read the file data 1 byte at a time until the requested number of bytes have been read or until the end-of-file is reached. To do so, set bit 7 of the `ioPosMode` field. Similarly, you can specify that `PBReadAsync` should stop reading data when it reaches an application-defined newline character. To do so, place the ASCII code of that character into the high-order byte of the `ioPosMode` field; you must also set bit 7 of that field to enable newline mode.

When reading data in newline mode, `PBReadAsync` returns the newline character as part of the data read and sets `ioActCount` to the actual number of bytes placed into the buffer (which includes the newline character).

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
`Files.h`

PBReadForkAsync

Reads data from an open fork.

```
void PBReadForkAsync (
    FSForkIOParam *paramBlock
);
```

Parameters

paramBlock
A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function. If there are fewer than `requestCount` bytes from the specified position to the logical end-of-file, then all of those bytes are read, and `eofErr` is returned.

`forkRefNum`
On input, the reference number of the fork to read from. You should have previously opened this fork using the `FSOpenFork` (page 514) call, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 740) and `PBOpenForkAsync` (page 739).

`positionMode`
On input, a constant specifying the base location within the fork for the start of the read. See "Position Mode Constants" (page 928) for a description of the constants which you can use to specify the base location. The caller can also use this parameter to hint to the File Manager whether the data being read should or should not be cached. Caching reads appropriately can be important in ensuring that your program access files efficiently. If you add the `forceReadMask` constant to the value you pass in this parameter, this tells the File Manager to force the data to be read directly from the disk. This is different from adding the `noCacheMask` constant since `forceReadMask` tells the File Manager to flush the appropriate part of the cache first, then ignore any data already in the cache. However, data that is read may be placed in the cache for future reads. The `forceReadMask` constant is also passed to the device driver, indicating that the driver should avoid reading from any device caches. See "Cache Constants" (page 889) for further description of the constants that you can use to indicate your preference for caching the read.

`positionOffset`
On input, the offset from the base location for the start of the read.

`requestCount`
: On input, the number of bytes to read. The value that you pass in this field should be greater than zero.

`buffer`
: A pointer to the buffer where the data will be returned.

`actualCount`
: On output, the number of bytes actually read. The value in this field should be equal to the value in the `requestCount` field unless there was an error during the read operation.

`PBReadForkAsync` reads data starting at the position specified by the `positionMode` and `positionOffset` fields. The function reads up to `requestCount` bytes into the buffer pointed to by the `buffer` field and sets the fork's current position to the byte immediately after the last byte read (that is, the initial position plus `actualCount`).

To verify that data previously written has been correctly transferred to disk, read it back in using the `forceReadMask` constant in the `positionMode` field and compare it with the data you previously wrote.

When reading data from a fork, it is important to pay attention to that way that your program accesses the fork, because this can have a significant performance impact. For best results, you should use an I/O size of at least 4KB and block align your read requests. In Mac OS X, you should align your requests to 4KB boundaries.

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

Declared In
`Files.h`

PBReadForkSync

Reads data from an open fork.

```
OSErr PBReadForkSync (
    FSForkIOParam *paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943). If there are fewer than `requestCount` bytes from the specified position to the logical end-of-file, then all of those bytes are read, and `eofErr` is returned.

Discussion
The relevant fields of the parameter block are:

`forkRefNum`
: On input, the reference number of the fork to read from. You should have previously opened this fork using the `FSOpenFork` (page 514) call, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 740) and `PBOpenForkAsync` (page 739).

`positionMode`
: On input, a constant specifying the base location within the fork for the start of the read. See "Position Mode Constants" (page 928) for a description of the constants which you can use to specify the base location. The caller can also use this parameter to hint to the File Manager whether the data being read should or should not be cached. Caching reads appropriately can be important in ensuring that your program access files efficiently. If you add the `forceReadMask` constant to the value you pass in this parameter, this tells the File Manager to force the data to be read directly from the disk. This is different from adding the `noCacheMask` constant since `forceReadMask` tells the File Manager to flush the appropriate part of the cache first, then ignore any data already in the cache. However, data that is read may be placed in the cache for future reads. The `forceReadMask` constant is also passed to the device driver, indicating that the driver should avoid reading from any device caches. See "Cache Constants" (page 889) for further description of the constants that you can use to indicate your preference for caching the read.

`positionOffset`
: On input, the offset from the base location for the start of the read.

`requestCount`
: On input, the number of bytes to read. The value that you pass in this field should be greater than zero.

`buffer`
: A pointer to the buffer where the data will be returned.

`actualCount`
: On output, the number of bytes actually read. The value in this field should be equal to the value in the `requestCount` field unless there was an error during the read operation.

`PBReadForkSync` reads data starting at the position specified by the `positionMode` and `positionOffset` fields. The function reads up to `requestCount` bytes into the buffer pointed to by the `buffer` field and sets the fork's current position to the byte immediately after the last byte read (that is, the initial position plus `actualCount`).

To verify that data previously written has been correctly transferred to disk, read it back in using the `forceReadMask` constant in the `positionMode` field and compare it with the data you previously wrote.

When reading data from a fork, it is important to pay attention to that way that your program accesses the fork, because this can have a significant performance impact. For best results, you should use an I/O size of at least 4KB and block align your read requests. In Mac OS X, you should align your requests to 4KB boundaries.

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

Declared In
`Files.h`

PBReadSync

Reads any number of bytes from an open file. (Deprecated in Mac OS X v10.5. Use `PBReadForkSync` (page 745) instead.)

```
OSErr PBReadSync (
    ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a basic File Manager parameter block.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioRefNum`

On input, a file reference number for an open file to be read.

`ioBuffer`

On input, a pointer to a data buffer into which the bytes are read.

`ioReqCount`

On input, the number of bytes requested. The value that you pass in this field should be greater than zero.

`ioActCount`

On output, the number of bytes actually read.

`ioPosMode`

On input, the positioning mode.

`ioPosOffset`

On input, the positioning offset. On output, the new position of the mark.

This function attempts to read `ioReqCount` bytes from the open file whose access path is specified in the `ioRefNum` field and transfer them to the data buffer pointed to by the `ioBuffer` field. The position of the mark is specified by `ioPosMode` and `ioPosOffset`. If your application tries to read past the logical end-of-file, `PBReadSync` reads the data, moves the mark to the end-of-file, and returns `eofErr` as its function result. Otherwise, `PBReadSync` moves the file mark to the byte following the last byte read and returns `noErr`.

You can specify that `PBReadSync` read the file data 1 byte at a time until the requested number of bytes have been read or until the end-of-file is reached. To do so, set bit 7 of the `ioPosMode` field. Similarly, you can specify that `PBReadSync` should stop reading data when it reaches an application-defined newline character. To do so, place the ASCII code of that character into the high-order byte of the `ioPosMode` field; you must also set bit 7 of that field to enable newline mode.

When reading data in newline mode, `PBReadSync` returns the newline character as part of the data read and sets `ioActCount` to the actual number of bytes placed into the buffer (which includes the newline character).

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

`Files.h`

PBRenameUnicodeAsync

Renames a file or folder.

```
void PBRenameUnicodeAsync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See FSRefParam (page 837) for a description of the FSRefParam data type.

Discussion

The relevant fields of the parameter block are:

ioCompletion

On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 794).

ioResult

On output, the result code of the function.

ref

On input, a pointer to an FSRef (page 837) for the file or directory to rename.

nameLength

On input, the length of the new name in Unicode characters.

name

On input, a pointer to the new Unicode name of the file or directory.

textEncodingHint

On input, the suggested text encoding to use when converting the Unicode name of the file or directory to some other encoding. If you pass the constant kTextEncodingUnknown, the File Manager will use a default value.

newRef

On output, a a pointer to the new FSRef for the file or directory. This field is optional; if you do not wish the FSRef returned, pass NULL.

Because renaming an object may change its FSRef, you should pass a non- NULL pointer in the newRef field and use the FSRef returned there to access the object after the renaming, if you wish to continue to refer to the object. The FSRef passed in the ref field may or may not be usable after the object is renamed. The FSRef returned in the newRef field may point to the same storage as the FSRef passed in ref.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Files.h

PBRenameUnicodeSync

Renames a file or folder.

```
OSErr PBRenameUnicodeSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 837) for a description of the `FSRefParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ref`

On input, a pointer to an `FSRef` (page 837) for the file or directory to rename.

`nameLength`

On input, the length of the new name in Unicode characters.

`name`

On input, a pointer to the new Unicode name of the file or directory.

`textEncodingHint`

On input, the suggested text encoding to use when converting the Unicode name of the file or directory to some other encoding. If you pass the constant `kTextEncodingUnknown`, the File Manager will use a default value.

`newRef`

On output, a a pointer to the new `FSRef` for the file or directory. This field is optional; if you do not wish the `FSRef` returned, pass `NULL`.

Because renaming an object may change its `FSRef`, you should pass a non- `NULL` pointer in the `newRef` field and use the `FSRef` returned there to access the object after the renaming, if you wish to continue to refer to the object. The `FSRef` passed in the `ref` field may or may not be usable after the object is renamed. The `FSRef` returned in the `newRef` field may point to the same storage as the `FSRef` passed in `ref`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBResolveFileIDRefAsync

Retrieves the filename and parent directory ID of a file given its file ID. (Deprecated in Mac OS X v10.5. Use `FSGetCatalogInfo` (page 494) instead.)

```
OSErr PBResolveFileIDRefAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to an `FIDParam` (page 818) variant of the HFS parameter block. See `HParamBlockRec` (page 857) for more information on the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
Most applications do not need to use this function. In general, you should track files using alias records, as described in the Alias Manager documentation. The Alias Manager uses file IDS internally as part of its search algorithms for finding the target of an alias record.

The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function. A return code of `fidNotFound` means that the specified file ID reference has become invalid, either because the file was deleted or because the file ID reference was destroyed by `PBDeleteFileIDRefSync` (page 597) or `PBDeleteFileIDRefAsync` (page 596).

`ioNamePtr`
: On input, a pointer to a pathname. If the name string is `NULL`, `PBResolveFileIDRefAsync` doe s not return the filename, but returns only the parent directory ID of the file in the `ioSrcDirID` field. If the name string is not `NULL` but is only a volume name, `PBResolveFileIDRefAsync` ignores the value in the `ioVRefNum` field and uses the volume name instead. On output, a pointer to the filename for the file with the given file ID.

`ioVRefNum`
: On input, a volume specification for the volume containing the file. This field can contain a volume reference number, a drive number, or 0 for the default volume.

`ioSrcDirID`
: On output, the file's parent directory ID.

`ioFileID`
: On input, a file ID for the file to retrieve information about.

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
`Files.h`

PBResolveFileIDRefSync

Retrieves the filename and parent directory ID of a file given its file ID. (Deprecated in Mac OS X v10.5. Use `FSGetCatalogInfo` (page 494) instead.)

```
OSErr PBResolveFileIDRefSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to an `FIDParam` (page 818) variant of the HFS parameter block. See `HParamBlockRec` (page 857) for more information on the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943). A return code of `fidNotFound` means that the specified file ID reference has become invalid, either because the file was deleted or because the file ID reference was destroyed by `PBDeleteFileIDRefSync` (page 597) or `PBDeleteFileIDRefAsync` (page 596).

Discussion

Most applications do not need to use this function. In general, you should track files using alias records, as described in the Alias Manager documentation. The Alias Manager uses file IDs internally as part of its search algorithms for finding the target of an alias record.

The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to a pathname. If the name string is `NULL`, `PBResolveFileIDRefSync` doe s not return the filename, but returns only the parent directory ID of the file in the `ioSrcDirID` field. If the name string is not `NULL` but is only a volume name, `PBResolveFileIDRefSync` ignores the value in the `ioVRefNum` field and uses the volume name instead. On output, a pointer to the filename of the file with the given file ID.

`ioVRefNum`

On input, a volume specification for the volume containing the file. This field can contain a volume reference number, drive number, or 0 for the default volume.

`ioSrcDirID`

On output, the file's parent directory ID.

`ioFileID`

On input, a file ID for the file to retrieve information about.

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

`Files.h`

PBSetCatalogInfoAsync

Sets the catalog information about a file or directory.

```
void PBSetCatalogInfoAsync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 837) for a description of the `FSRefParam` data type.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function.

`ref`

On input, a pointer to an `FSRef` (page 837) specifying the file or directory whose information is to be changed.

`whichInfo`

On input, a bitmap specifying which catalog information fields to set. Only some of the catalog information fields may be set. These fields are given by the constant `kFSCatInfoSettableInfo`; no other bits may be set in the `whichInfo` field. See "Catalog Information Bitmap Constants" (page 891) for a description of the bits in this field.

To set the user ID (UID) and group ID (GID), specify the `kFSCatInfoSetOwnership` flag in this field. The File Manager attempts to set the user and group ID to the values specified in the `permissions` field of the catalog information structure. If `PBSetCatalogInfoAsync` cannot set the user and group IDs, it returns an error.

`catInfo`

On input, a pointer to the `FSCatalogInfo` (page 826) structure containing the new catalog information. Only some of the catalog information fields may be set. The fields which may be set are:

- `createDate`
- `contentModDate`
- `attributeModDate`
- `accessDate`
- `backupDate`
- `permissions`
- `finderInfo`
- `extFinderInfo`
- `textEncodingHint`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBSetCatalogInfoSync

Sets the catalog information about a file or directory.

```
OSErr PBSetCatalogInfoSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 837) fro s description of the `FSRefParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ref`

On input, a pointer to an `FSRef` (page 837) specifying the file or directory whose information is to be changed.

`whichInfo`

On input, a bitmap specifying which catalog information fields to set. Only some of the catalog information fields may be set. These fields are given by the constant `kFSCatInfoSettableInfo`; no other bits may be set in the `whichInfo` field. See "Catalog Information Bitmap Constants" (page 891) for a description of the bits in this field.

To set the user ID (UID) and group ID (GID), specify the `kFSCatInfoSetOwnership` flag in this field. The File Manager attempts to set the user and group ID to the values specified in the `permissions` field of the catalog information structure. If `PBSetCatalogInfoSync` cannot set the user and group IDs, it returns an error.

`catInfo`

On input, a pointer to the `FSCatalogInfo` (page 826) structure containing the new catalog information. Only some of the catalog information fields may be set. The fields which may be set are:

- `createDate`
- `contentModDate`
- `attributeModDate`
- `accessDate`
- `backupDate`
- `permissions`
- `finderInfo`
- `extFinderInfo`
- `textEncodingHint`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBSetCatInfoAsync

Modifies catalog information for a file or directory. (Deprecated in Mac OS X v10.4. Use `PBSetCatalogInfoAsync` (page 751) instead.)

```
OSErr PBSetCatInfoAsync (
   CInfoPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to an HFS catalog information parameter block. See `CInfoPBRec` (page 802) for a description of the `CInfoPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The `PBSetCatInfoAsync` function sets information about a file or directory. When used to set information about a file, it works much as `PBHSetFInfoAsync` (page 721) does, but lets you set some additional information.

If the object is a file, the relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to a pathname.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioFlFndrInfo`
On input, Finder information for the file.

`ioDirID`
On input, the parent directory ID of the file.

`ioFlCrDat`
On input, the date and time of the file's creation.

`ioFlMdDat`
On input, the date and time of the file's last modification.

`ioFlBkDat`
On input, the date and time of the file's last backup.

`ioFlXFndrInfo`
On input, extended Finder information.

If the object is a directory, the relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function.

`ioNamePtr`
: On input, a pointer to a pathname.

`ioVRefNum`
: On input, a volume reference number, drive number, or 0 for the default volume.

`ioDrUsrWds`
: On input, information used by the Finder.

`ioDrDirID`
: On input, the directory ID.

`ioDrCrDat`
: On input, the date and time of the directory's creation.

`ioDrMdDat`
: On input, the date and time of the directory's last modification.

`ioDrBkDat`
: On input, the date and time of the directory's last backup.

`ioDrFndrInfo`
: On input, additional information used by the Finder.

To modify the catalog information for a named fork other than the data and resource fork, or to modify other catalog information maintained on HFS Plus volumes that is not modifiable through `PBSetCatInfoAsync`, use one of the functions, `FSSetCatalogInfo` (page 540) , `PBSetCatalogInfoSync` (page 753) , or `PBSetCatalogInfoAsync` (page 751).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBSetCatInfoSync

Modifies catalog information for a file or directory. (Deprecated in Mac OS X v10.4. Use `PBSetCatalogInfoSync` (page 753) instead.)

```
OSErr PBSetCatInfoSync (
   CInfoPBPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to an HFS catalog information parameter block. See `CInfoPBRec` (page 802) for a description of the `CInfoPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The `PBSetCatInfoSync` function sets information about a file or directory. When used to set information about a file, it works much as `PBHSetFInfoSync` (page 722) does, but lets you set some additional information.

If the object is a file, the relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a pathname.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioFlFndrInfo`
On input, Finder information for the file.

`ioDirID`
On input, the parent directory ID of the file.

`ioFlCrDat`
On input, the date and time of the file's creation.

`ioFlMdDat`
On input, the date and time of the file's last modification.

`ioFlBkDat`
On input, the date and time of the file's last backup.

`ioFlXFndrInfo`
On input, extended Finder information.

If the object is a directory, the relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a pathname.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioDrUsrWds`
On input, information used by the Finder.

`ioDrDirID`
On input, the directory ID.

`ioDrCrDat`
On input, the date and time of the directory's creation.

`ioDrMdDat`
On input, the date and time of the directory's last modification.

`ioDrBkDat`
On input, the date and time of the directory's last backup.

`ioDrFndrInfo`
On input, additional information used by the Finder.

To modify the catalog information for a named fork other than the data and resource fork, or to modify other catalog information maintained on HFS Plus volumes that is not modifiable through `PBSetCatInfoSync`, use one of the functions, `FSSetCatalogInfo` (page 540) , `PBSetCatalogInfoSync` (page 753) , or `PBSetCatalogInfoAsync` (page 751).

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

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBSetEOFAsync

Sets the logical size of an open file. (Deprecated in Mac OS X v10.4. Use `PBSetForkSizeAsync` (page 761) instead.)

```
OSErr PBSetEOFAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the `IOParam` (page 862) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 866) for a description of the `ParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function.

`ioRefNum`
On input, a file reference number for the open file.

`ioMisc`
On input, the new logical size (the logical end-of-file) of the given file. Because the `ioMisc` field is of type `Ptr`, you must coerce the desired value from a long integer to type `Ptr`. If the value of the `ioMisc` field is 0, all space occupied by the file on the volume is released. The file still exists, but it contains 0 bytes. Setting a file fork's end-of-file to 0 is therefore not the same as deleting the file, which removes both file forks at once.

If you attempt to set the logical end-of-file beyond the current physical end-of-file, another allocation block is added to the file if there isn't enough space on the volume, no change is made and `PBSetEOFAsync` returns `dskFulErr` as its function result.

To ensure that your changes to the file are written to disk, call one of the functions, `FlushVol` (page 466) , `PBFlushVolSync` (page 641) , or `PBFlushVolAsync` (page 640). To set the size of a named fork other than the data and resource forks, or to grow the size of a file beyond 2GB, you must use the `FSSetForkSize` (page 542) function, or one of the corresponding parameter block calls, `PBSetForkSizeSync` (page 762) and `PBSetForkSizeAsync` (page 761).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Files.h`

PBSetEOFSync

Sets the logical size of an open file. (Deprecated in Mac OS X v10.4. Use `PBSetForkSizeSync` (page 762) instead.)

```
OSErr PBSetEOFSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `IOParam` (page 862) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 866) for a description of the `ParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioRefNum`

On input, a file reference number for the open file.

`ioMisc`

On input, the new logical size (the logical end-of-file) of the given file. Because the `ioMisc` field is of type `Ptr`, you must coerce the desired value from a long integer to type `Ptr`. If the value of the `ioMisc` field is 0, all space occupied by the file on the volume is released. The file still exists, but it contains 0 bytes. Setting a file fork's end-of-file to 0 is therefore not the same as deleting the file, which removes both file forks at once.

If you attempt to set the logical end-of-file beyond the current physical end-of-file, another allocation block is added to the file if there isn't enough space on the volume, no change is made and `PBSetEOFSync` returns `dskFulErr` as its function result.

To ensure that your changes to the file are written to disk, call one of the functions, `FlushVol` (page 466) , `PBFlushVolSync` (page 641) , or `PBFlushVolAsync` (page 640). To set the size of a named fork other than the data and resource forks, or to grow the size of a file beyond 2GB, you must use the `FSSetForkSize` (page 542) function, or one of the corresponding parameter block calls, `PBSetForkSizeSync` (page 762) and `PBSetForkSizeAsync` (page 761).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBSetForeignPrivsAsync

Changes the native access-control information for a file or directory stored on a volume managed by a foreign file system. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBSetForeignPrivsAsync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented 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

`Files.h`

PBSetForeignPrivsSync

Changes the native access-control information for a file or directory stored on a volume managed by a foreign file system. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBSetForeignPrivsSync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented 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

`Files.h`

PBSetForkPositionAsync

Sets the current position of an open fork.

```
void PBSetForkPositionAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function. This function returns the result code `posErr` if you attempt to set the current position of the fork to an offset before the start of the file.

`forkRefNum`
On input, the reference number of a fork previously opened by the `FSOpenFork` (page 514) , `PBOpenForkSync` (page 740) , or `PBOpenForkAsync` (page 739) function.

`positionMode`
On input, a constant specifying the base location within the fork for the new position. If this field is equal to `fsAtMark`, then the `positionOffset` field is ignored. See "Position Mode Constants" (page 928) for a description of the constants you can use to specify the base location.

`positionOffset`
On input, the offset of the new position from the base location specified in the `positionMode` field.

Special Considerations

To determine if the `PBSetForkPositionAsync` function is present, call the `Gestalt` function with the `gestaltFSAttr` selector. If the `PBSetForkPositionAsync` function is present, but the volume does not directly support it, the File Manager will automatically call the `PBSetFPosAsync` (page 763) function. However, if the volume does not directly support the `PBSetForkPositionAsync` function, you can only set the file position for the data and resource forks, and you cannot grow these files beyond 2GB.

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

Declared In
`Files.h`

PBSetForkPositionSync

Sets the current position of an open fork.

```
OSErr PBSetForkPositionSync (
   FSForkIOParam *paramBlock
);
```

Parameters

`paramBlock`
A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943). This function returns the result code `posErr` if you attempt to set the current position of the fork to an offset before the start of the file.

Discussion
The relevant fields of the parameter block are:

`forkRefNum`
: On input, the reference number of a fork previously opened by the `FSOpenFork` (page 514) , `PBOpenForkSync` (page 740) , or `PBOpenForkAsync` (page 739) function.

`positionMode`
: On input, a constant specifying the base location within the fork for the new position. If this field is equal to `fsAtMark`, then the `positionOffset` field is ignored. See "Position Mode Constants" (page 928) for a description of the constants you can use to specify the base location.

`positionOffset`
: On input, the offset of the new position from the base location specified in the `positionMode` field.

Special Considerations

To determine if the `PBSetForkPositionSync` function is present, call the `Gestalt` function with the `gestaltFSAttr` selector. If the `PBSetForkPositionSync` function is present, but the volume does not directly support it, the File Manager will automatically call the `PBSetFPosSync` (page 764) function. However, if the volume does not directly support the `PBSetForkPositionSync` function, you can only set the file position for the data and resource forks, and you cannot grow these files beyond 2GB.

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

Declared In
`Files.h`

PBSetForkSizeAsync

Changes the size of an open fork.

```
void PBSetForkSizeAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function. If there is not enough space on the volume to extend the fork, then `dskFulErr` is returned and the fork's size is unchanged.

`forkRefNum`
: On input, the reference number of the open fork. You can obtain a fork reference number with the `FSOpenFork` (page 514) function, or with one of the corresponding parameter block calls, `PBOpenForkSync` (page 740) and `PBOpenForkAsync` (page 739).

`positionMode`
: On input, a constant indicating the base location within the fork for the new size. See "Position Mode Constants" (page 928) for more information about the constants you can use to specify the base location.

`positionOffset`
: On input, the offset of the new size from the base location specified in the `positionMode` field.

The `PBSetForkSizeAsync` function sets the logical end-of-file to the position indicated by the `positionMode` and `positionOffset` fields. The fork's new size may be less than, equal to, or greater than the fork's current size. If the fork's new size is greater than the fork's current size, then the additional bytes, between the old and new size, will have an undetermined value.

If the fork's current position is larger than the fork's new size, then the current position will be set to the new fork size. That is, the current position will be equal to the logical end of file.

Special Considerations

You do not need to check that the volume supports the `PBSetForkSizeAsync` function. If a volume does not support the `PBSetForkSizeAsync` function, but the `PBSetForkSizeAsync` function is present, the File Manager automatically calls the `PBSetEOFAsync` (page 757) function and translates between the calls appropriately.

Note, however, that if the volume does not support the `PBSetForkSizeAsync` function, you can only access the data and resource forks, and you cannot grow the fork beyond 2GB. To check that the `PBSetForkSizeAsync` function is present, call the `Gestalt` function.

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

Declared In
`Files.h`

PBSetForkSizeSync

Changes the size of an open fork.

```
OSErr PBSetForkSizeSync (
   FSForkIOParam *paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943). If there is not enough space on the volume to extend the fork, then `dskFulErr` is returned and the fork's size is unchanged.

Discussion
The relevant fields of the parameter block are:

`forkRefNum`
: On input, the reference number of the open fork. You can obtain a fork reference number with the `FSOpenFork` (page 514) function, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 740) and `PBOpenForkAsync` (page 739).

`positionMode`
: On input, a constant indicating the base location within the fork for the new size. See "Position Mode Constants" (page 928) for more information about the constants you can use to specify the base location.

`positionOffset`
: On input, the offset of the new size from the base location specified in the `positionMode` field.

The `PBSetForkSizeSync` function sets the logical end-of-file to the position indicated by the `positionMode` and `positionOffset` fields. The fork's new size may be less than, equal to, or greater than the fork's current size. If the fork's new size is greater than the fork's current size, then the additional bytes, between the old and new size, will have an undetermined value.

If the fork's current position is larger than the fork's new size, then the current position will be set to the new fork size. That is, the current position will be equal to the logical end-of-file.

Special Considerations

You do not need to check that the volume supports the `PBSetForkSizeSync` function. If a volume does not support the `PBSetForkSizeSync` function, but the `PBSetForkSizeSync` function is present, the File Manager automatically calls the `PBSetEOFSync` (page 758) function and translates between the calls appropriately.

Note, however, that if the volume does not support the `PBSetForkSizeSync` function, you can only access the data and resource forks, and you cannot grow the fork beyond 2GB. To check that the `PBSetForkSizeSync` function is present, call the `Gestalt` function.

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

Declared In
`Files.h`

PBSetFPosAsync

Sets the position of the file mark. (Deprecated in Mac OS X v10.4. Use `PBSetForkPositionAsync` (page 759) instead.)

```
OSErr PBSetFPosAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `IOParam` (page 862) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 866) for a description of the `ParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function.

`ioRefNum`
: On input, the file reference number for an open file.

`ioPosMode`
: On input, a constant indicating how to position the mark; this field must contain one of the values described in "Position Mode Constants" (page 928).

`ioPosOffset`
: On input, the offset from the base location specified by the `ioPosMode` field for the file mark. If you specify `fsAtMark` in the `ioPosMode` field, the mark is left wherever it's currently positioned and the value in the `ioPosOffset` field is ignored. If you specify `fsFromLEOF`, the value in `ioPosOffset` must be less than or equal to 0. On output, the position at which the mark was actually set.

The `PBSetFPosAsync` function sets the mark of the specified file to the position specified by the `ioPosMode` and `ioPosOffset` fields. If you try to set the mark past the logical end-of-file, `PBSetFPosAsync` moves the mark to the end-of-file and returns `eofErr` as its function result.

To set the file mark position for a named fork other than the data and resource forks, or to position the file mark at a point more than 2GB into the file, use the `FSSetForkPosition` (page 541) function, or one of the corresponding parameter block calls, `PBSetForkPositionSync` (page 760) and `PBSetForkPositionAsync` (page 759).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBSetFPosSync

Sets the position of the file mark. (Deprecated in Mac OS X v10.4. Use `PBSetForkPositionSync` (page 760) instead.)

```
OSErr PBSetFPosSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock
: A pointer to the `IOParam` (page 862) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 866) for a description of the `ParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioRefNum`
: On input, the file reference number for an open file.

`ioPosMode`
: On input, a constant indicating how to position the file mark; this field must contain one of the values described in "Position Mode Constants" (page 928).

`ioPosOffset`
: On input, the offset from the base location specified by the `ioPosMode` field for the file mark. If you specify `fsAtMark` in the `ioPosMode` field, the mark is left wherever it's currently positioned and the value in the `ioPosOffset` field is ignored. If you specify `fsFromLEOF`, the value in `ioPosOffset` must be less than or equal to 0. On output, the position at which the mark was actually set.

The `PBSetFPosSync` function sets the mark of the specified file to the position specified by the `ioPosMode` and `ioPosOffset` fields. If you try to set the mark past the logical end-of-file, `PBSetFPosSync` moves the mark to the end-of-file and returns `eofErr` as its function result.

To set the file mark position for a named fork other than the data and resource forks, or to position the file mark at a point more than 2GB into the file, use the `FSSetForkPosition` (page 541) function, or one of the corresponding parameter block calls, `PBSetForkPositionSync` (page 760) and `PBSetForkPositionAsync` (page 759).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBSetVInfoAsync

Changes information about a volume. (Deprecated in Mac OS X v10.4. Use `PBSetVolumeInfoAsync` (page 767) instead.)

```
OSErr PBSetVInfoAsync (
   HParmBlkPtr paramBlock
);
```

Parameters
`paramBlock`
: A pointer to the `HVolumeParam` (page 859) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function.

`ioNamePtr`
On input, a pointer to the volume's name. You can specify a new name for the volume here. You cannot specify the volume by name you must use either the volume reference number or the drive number.

`ioVRefNum`
On input, a volume reference number or drive number for the volume whose information is to be changed; or 0 for the default volume.

`ioVCrDate`
On input, the date and time of the volume's initialization.

`ioVLsMod`
On input, the date and time of the volume's last modification.

`ioVAtrb`
On input, the volume attributes. Only bit 15 of the `ioVAtrb` field can be changed; setting it locks the volume. See "Volume Information Attribute Constants" (page 937) for a description of the volume attributes.

`ioVBkUp`
On input, the date and time of the volume's last backup.

`ioVSeqNum`
Used internally.

`ioVFndrInfo`
On input, Finder information for the volume.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBSetVInfoSync

Changes information about a volume. (Deprecated in Mac OS X v10.4. Use `PBSetVolumeInfoSync` (page 768) instead.)

```
OSErr PBSetVInfoSync (
   HParmBlkPtr paramBlock
);
```

Parameters
`paramBlock`
A pointer to the `HVolumeParam` (page 859) variant of the basic HFS parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the volume's name. You can specify a new name for the volume here. You cannot specify the volume by name you must use either the volume reference number or the drive number.

`ioVRefNum`
On input, a volume reference number or drive number for the volume whose information is to be changed; or 0 for the default volume.

`ioVCrDate`
On input, the date and time of the volume's initialization.

`ioVLsMod`
On input, the date and time of the volume's last modification.

`ioVAtrb`
On input, the volume attributes. Only bit 15 of the `ioVAtrb` field can be changed; setting it locks the volume. See "Volume Information Attribute Constants" (page 937) for a description of the volume attributes.

`ioVBkUp`
On input, the date and time of the volume's last backup.

`ioVSeqNum`
Used internally.

`ioVFndrInfo`
On input, Finder information for the volume.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBSetVolumeInfoAsync

Sets information about a volume.

```
void PBSetVolumeInfoAsync (
   FSVolumeInfoParam *paramBlock
);
```

Parameters

`paramBlock`
A pointer to a volume information parameter block. See `FSVolumeInfoParam` (page 845) for a description of the `FSVolumeInfoParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
On output, the result code of the function.

`ioVRefNum`
On input, the volume reference number of the volume whose information is to be changed.

`whichInfo`
On input, a bitmap specifying which information to set. Only some of the volume information fields may be set. The settable fields are given by the constant `kFSVolInfoSettableInfo`; no other bits may be set in `whichInfo`. The fields which may be set are the backupDate, finderInfo, and `flags` fields. See "Volume Information Bitmap Constants" (page 938) for a description of the bits in this parameter.

`volumeInfo`
On input, the new volume information. See `FSVolumeInfo` (page 842) for more information about the volume information structure.

To set information about the root directory of a volume, use the `FSSetCatalogInfo` (page 540) function, or one of the corresponding parameter block calls, `PBSetCatalogInfoSync` (page 753) and `PBSetCatalogInfoAsync` (page 751).

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

Declared In
`Files.h`

PBSetVolumeInfoSync

Sets information about a volume.

```
OSErr PBSetVolumeInfoSync (
   FSVolumeInfoParam *paramBlock
);
```

Parameters

`paramBlock`
A pointer to a volume information parameter block. See `FSVolumeInfoParam` (page 845) for a description of the `FSVolumeInfoParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioVRefNum`
On input, the volume reference number of the volume whose information is to be changed.

`whichInfo`
On input, a bitmap specifying which information to set. Only some of the volume information fields may be set. The settable fields are given by the constant `kFSVolInfoSettableInfo`; no other bits may be set in `whichInfo`. The fields which may be set are the backupDate, finderInfo, and `flags` fields. See "Volume Information Bitmap Constants" (page 938) for a description of the bits in this parameter.

`volumeInfo`

On input, the new volume information. See `FSVolumeInfo` (page 842) for more information about the volume information structure.

To set information about the root directory of a volume, use the `FSSetCatalogInfo` (page 540) function, or one of the corresponding parameter block calls, `PBSetCatalogInfoSync` (page 753) and `PBSetCatalogInfoAsync` (page 751).

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

Declared In
`Files.h`

PBShareAsync

Establishes a local volume or directory as a share point. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBShareAsync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented 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
`Files.h`

PBShareSync

Establishes a local volume or directory as a share point. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBShareSync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented 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
`Files.h`

PBUnlockRangeAsync

Unlocks a portion of a file. (Deprecated in Mac OS X v10.4. Use `PBXUnlockRangeAsync` (page 785) instead.)

```
OSErr PBUnlockRangeAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `IOParam` (page 862) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 866) for a description of the `ParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the result code of the function. If you call `PBUnlockRangeAsync` on a file system that does not implement it—for example, SMB—`PBUnlockRangeAsync` returns `noErr` and does nothing.

`ioRefNum`

On input, the file reference number of the file owning the range to unlock.

`ioReqCount`

On input, the number of bytes in the range.

`ioPosMode`

On input, a constant specifying the base location for the start of the locked range. See "Position Mode Constants" (page 928) for more information on the constants you can use to indicate the base location.

`ioPosOffset`

On input, the offset from the base location specified in the `ioPosMode` field for the start of the locked range.

The `PBUnlockRangeAsync` function unlocks a portion of a file that you locked with `PBLockRangeSync` (page 730) or `PBLockRangeAsync` (page 728). The beginning of the range to be unlocked is determined by the `ioPosMode` and `ioPosOffset` fields. The end of the range to be unlocked is determined by the beginning of the range and the `ioReqCount` field. For example, to unlock the first 50 bytes in a file, set `ioReqCount` to 50, `ioPosMode` to `fsFromStart`, and `ioPosOffset` to 0. The range of bytes to be unlocked must be the exact same range locked by a previous call to `PBLockRangeSync` (page 730) or `PBLockRangeAsync` (page 728).

If for some reason you need to unlock a range whose beginning or length is unknown, you can simply close the file. When a file is closed, all locked ranges held by the user are unlocked.

Special Considerations

The `PBUnlockRangeAsync` function does nothing if the file specified in the `ioRefNum` field is open with shared read/write permission but is not located on a remote server volume or is not located under a share point on a local volume. To check whether file sharing is currently on, check that the `bHasPersonalAccessPrivileges` bit in the `vMAttrib` field of the `GetVolParmsInfoBuffer` (page 847) returned by the `PBHGetVolParmsSync` (page 695) function is set.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBUnlockRangeSync

Unlocks a portion of a file. (Deprecated in Mac OS X v10.4. Use `PBXUnlockRangeSync` (page 786) or `FSUnlockRange` (page 543) instead.)

```
OSErr PBUnlockRangeSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the `IOParam` (page 862) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 866) for a description of the `ParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943). If you call `PBUnlockRangeSync` on a file system that does not implement it—for example, SMB—`PBUnlockRangeSync` returns `noErr` and does nothing.

Discussion
The relevant fields of the parameter block are:

`ioRefNum`
On input, the file reference number of the file owning the range to unlock.

`ioReqCount`
On input, the number of bytes in the range.

`ioPosMode`
On input, a constant specifying the base location for the start of the locked range. See "Position Mode Constants" (page 928) for more information on the constants you can use to indicate the base location.

`ioPosOffset`
On input, the offset from the base location specified in the `ioPosMode` field for the start of the locked range.

The `PBUnlockRangeSync` function unlocks a portion of a file that you locked with `PBLockRangeSync` (page 730) or `PBLockRangeAsync` (page 728). The beginning of the range to be unlocked is determined by the `ioPosMode` and `ioPosOffset` fields. The end of the range to be unlocked is determined by the beginning of the range and the `ioReqCount` field. For example, to unlock the first 50 bytes in a file, set `ioReqCount` to 50, `ioPosMode` to `fsFromStart`, and `ioPosOffset` to 0. The range of bytes to be unlocked must be the exact same range locked by a previous call to `PBLockRangeSync` (page 730) or `PBLockRangeAsync` (page 728).

If for some reason you need to unlock a range whose beginning or length is unknown, you can simply close the file. When a file is closed, all locked ranges held by the user are unlocked.

Special Considerations

The `PBUnlockRangeSync` function does nothing if the file specified in the `ioRefNum` field is open with shared read/write permission but is not located on a remote server volume or is not located under a share point on a local volume. To check whether file sharing is currently on, check that the `bHasPersonalAccessPrivileges` bit in the `vMAttrib` field of the `GetVolParmsInfoBuffer` (page 847) returned by the `PBHGetVolParmsSync` (page 695) function is set.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBUnmountVol

Unmounts a volume. (Deprecated in Mac OS X v10.4. Use `FSEjectVolumeSync` (page 486) or `FSUnmountVolumeSync` (page 545) instead.)

```
OSErr PBUnmountVol (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `VolumeParam` (page 873) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 866) for a description of the `ParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioResult`

On output, the result code of the function.

`ioNamePtr`

On input, a pointer to the name of the volume.

`ioVRefNum`

On input, the volume reference number of the volume to unmount, or 0 for the default volume.

This function calls `PBFlushVolSync` to flush the specified volume, unmounts and ejects the volume, and releases the memory used for the volume. Prior to calling this function, all user files on the volume must be closed. Ejecting a volume results in the unmounting of other volumes on the same device.

The `PBUnmountVol` function always executes synchronously.

Special Considerations

Don't unmount the startup volume. Doing so will cause a system crash.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBUnshareAsync

Makes a share point unavailable on the network. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBUnshareAsync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented 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
`Files.h`

PBUnshareSync

Makes a share point unavailable on the network. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBUnshareSync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented 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
`Files.h`

PBVolumeMount

Mounts a volume. (Deprecated in Mac OS X v10.5. Use `FSVolumeMount` (page 545) instead.)

```
OSErr PBVolumeMount (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `IOParam` (page 862) variant of the basic File Manager parameter block. See `HParamBlockRec` (page 857) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioVRefNum`

On output, a volume reference number for the mounted volume.

`ioBuffer`

On input, a pointer to mounting information. You can use the volume mounting information returned by the `PBGetVolMountInfo` (page 668) function or you can use a volume mounting information structure filled in by your application. If you're mounting an AppleShare volume, place the volume's AFP mounting information structure in the buffer pointed to by the `ioBuffer` field.

This function allows your application to record the mounting information for a volume and then to mount the volume later.

The `PBGetVolMountInfo` function does not return the user and volume passwords they're returned blank. Typically, your application asks the user for any necessary passwords and fills in those fields just before calling `PBVolumeMount`. If you want to mount a volume with guest status, pass an empty string as the user password.

If you have enough information about the volume, you can fill in the mounting structure yourself and call `PBVolumeMount`, even if you did not save the mounting information while the volume was mounted. To mount an AFP volume, you must fill in the structure with at least the zone name, server name, user name, user password, and volume password. You can lay out the fields in any order within the data field, as long as you specify the correct offsets.

In general, it is easier to mount remote volumes by creating and then resolving alias records that describe those volumes. The Alias Manager displays the standard user interface for user authentication when resolving alias records for remote volumes. As a result, this function is primarily of interest for applications that need to mount remote volumes with no user interface or with some custom user interface.

Special Considerations

AFP volumes currently ignore the user authentication method passed in the `uamType` field of the volume mounting information structure whose address is passed in the `ioBuffer` field of the parameter block. The most secure available method is used by default, except when a user mounts the volume as Guest and uses the `kNoUserAuthentication` authentication method.

This function executes synchronously. You should not call it at interrupt time.

Version Notes
The File Sharing workstation software introduced in system software version 7.0 does not currently pass the volume password. The AppleShare 3.0 workstation software does, however, pass the volume password.

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
`Files.h`

PBWaitIOComplete

Keeps the system idle until either an interrupt occurs or the specified timeout value is reached. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
OSErr PBWaitIOComplete (
    ParmBlkPtr paramBlock,
    Duration timeout
);
```

Parameters

paramBlock
A pointer to a basic File Manager parameter block.

timeout
The maximum length of time you want the system to be kept idle.

Return Value
A result code. If the timeout value is reached, returns `kMPTimeoutErr`.

Special Considerations

This function is not implemented in Mac OS X.

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
`Files.h`

PBWriteAsync

Writes any number of bytes to an open file. (Deprecated in Mac OS X v10.5. Use `PBWriteForkAsync` (page 776) instead.)

```
OSErr PBWriteAsync (
    ParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to a basic File Manager parameter block.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine.

`ioResult`
On output, the result code of the function.

`ioRefNum`
On input, a file reference number for the open file to which to write.

`ioBuffer`
On input, a pointer to a data buffer containing the bytes to write.

`ioReqCount`
On input, the number of bytes requested.

`ioActCount`
On output, the number of bytes actually written.

`ioPosMode`
On input, the positioning mode.

`ioPosOffset`
On input, the positioning offset. On output, the new position of the mark.

The `PBWriteAsync` function takes `ioReqCount` bytes from the buffer pointed to by `ioBuffer` and attempts to write them to the open file whose access path is specified by `ioRefNum`. The position of the mark is specified by `ioPosMode` and `ioPosOffset`. If the write operation completes successfully, `PBWriteAsync` moves the file mark to the byte following the last byte written and returns `noErr`. If you try to write past the logical end-of-file, `PBWriteAsync` moves the logical end-of-file. If you try to write past the physical end-of-file, `PBWriteAsync` adds one or more clumps to the file and moves the physical end-of-file accordingly.

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
`Files.h`

PBWriteForkAsync

Writes data to an open fork.

```
void PBWriteForkAsync (
    FSForkIOParam *paramBlock
);
```

Parameters

`paramBlock`
A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam`.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`
: On output, the result code of the function. If there is not enough space on the volume to write `requestCount` bytes, then `dskFulErr` is returned.

`forkRefNum`
: On input, the reference number of the fork to which to write. You should have previously opened the fork using the `FSOpenFork` (page 514) function, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 740) and `PBOpenForkAsync` (page 739).

`positionMode`
: On input, a constant specifying the base location within the fork for the start of the write. See "Position Mode Constants" (page 928) for a description of the constants which you can use to specify the base location. The caller can also use this parameter to hint to the File Manager whether the data being written should or should not be cached. See "Cache Constants" (page 889) for further description of the constants that you can use to indicate your preference for caching.

`positionOffset`
: On input, the offset from the base location for the start of the write.

`requestCount`
: On input, the number of bytes to write.

`buffer`
: A pointer to a buffer containing the data to write.

`actualCount`
: On output, the number of bytes actually written. The value in the `actualCount` field will be equal to the value in the `requestCount` field unless there was an error during the write operation.

`PBWriteForkAsync` writes data starting at the position specified by the `positionMode` and `positionOffset` fields. The function attempts to write `requestCount` bytes from the buffer pointed to by the `buffer` field and sets the fork's current position to the byte immediately after the last byte written (that is, the initial position plus `actualCount`).

When writing data to a fork, it is important to pay attention to that way that your program accesses the fork, because this can have a significant performance impact. For best results, you should use an I/O size of at least 4KB and block align your write requests. In Mac OS X, you should align your requests to 4KB boundaries.

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

Declared In
`Files.h`

PBWriteForkSync

Writes data to an open fork.

```
OSErr PBWriteForkSync (
    FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 833) for a description of the `FSForkIOParam`.

Return Value

A result code. See "File Manager Result Codes" (page 943). If there is not enough space on the volume to write `requestCount` bytes, then `dskFulErr` is returned.

Discussion

The relevant fields of the parameter block are:

`ioResult`

On output, the result code of the function. If there is not enough space on the volume to write `requestCount` bytes, then `dskFulErr` is returned.

`forkRefNum`

On input, the reference number of the fork to which to write. You should have previously opened the fork using the `FSOpenFork` (page 514) function, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 740) and `PBOpenForkAsync` (page 739).

`positionMode`

On input, a constant specifying the base location within the fork for the start of the write. See "Position Mode Constants" (page 928) for a description of the constants which you can use to specify the base location. The caller can also use this parameter to hint to the File Manager whether the data being written should or should not be cached. See "Cache Constants" (page 889) for further description of the constants that you can use to indicate your preference for caching.

`positionOffset`

On input, the offset from the base location for the start of the write.

`requestCount`

On input, the number of bytes to write.

`buffer`

A pointer to a buffer containing the data to write.

`actualCount`

On output, the number of bytes actually written. The value in the `actualCount` field will be equal to the value in the `requestCount` field unless there was an error during the write operation.

`PBWriteForkSync` writes data starting at the position specified by the `positionMode` and `positionOffset` fields. The function attempts to write `requestCount` bytes from the buffer pointed to by the `buffer` field and sets the fork's current position to the byte immediately after the last byte written (that is, the initial position plus `actualCount`).

When writing data to a fork, it is important to pay attention to that way that your program accesses the fork, because this can have a significant performance impact. For best results, you should use an I/O size of at least 4KB and block align your write requests. In Mac OS X, you should align your requests to 4KB boundaries.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBWriteSync

Writes any number of bytes to an open file. (Deprecated in Mac OS X v10.5. Use `PBWriteForkSync` (page 777) instead.)

```
OSErr PBWriteSync (
    ParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to a basic File Manager parameter block.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioRefNum`
On input, a file reference number for the open file to which to write.

`ioBuffer`
On input, a pointer to a data buffer containing the bytes to write.

`ioReqCount`
On input, the number of bytes requested.

`ioActCount`
On output, the number of bytes actually written.

`ioPosMode`
On input, the positioning mode.

`ioPosOffset`
On input, the positioning offset. On output, the new position of the mark.

The `PBWriteSync` function takes `ioReqCount` bytes from the buffer pointed to by `ioBuffer` and attempts to write them to the open file whose access path is specified by `ioRefNum`. The position of the mark is specified by `ioPosMode` and `ioPosOffset`. If the write operation completes successfully, `PBWriteSync` moves the file mark to the byte following the last byte written and returns `noErr`. If you try to write past the logical end-of-file, `PBWriteSync` moves the logical end-of-file. If you try to write past the physical end-of-file, `PBWriteSync` adds one or more clumps to the file and moves the physical end-of-file accordingly.

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

`Files.h`

PBXGetVolInfoAsync

Returns information about a volume, including size information for volumes up to 2 terabytes. (Deprecated in Mac OS X v10.4. Use `FSGetVolumeInfo` (page 500) instead.)

```
OSErr PBXGetVolInfoAsync (
   XVolumeParamPtr paramBlock
);
```

Parameters

paramBlock

A pointer to an extended volume parameter block. See `XVolumeParam` (page 882) for a description of the `XVolumeParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 794).

`ioResult`

On output, the function result.

`ioNamePtr`

On input, a pointer to a buffer. You should pass a pointer to a `Str31` value if you want the volume name returned; otherwise, pass `NULL`. If you specify a negative number in the `ioVolIndex` field, this buffer should hold the name of the volume for which to return information. On output, a pointer to the volume's name.

`ioVRefNum`

On input, a volume reference number, drive number, or 0 for the default volume. If the value in the `ioVolIndex` field is negative, the File Manager uses the name in the `ioNamePtr` field, along with the value in the `ioVRefNum` field, to determine the volume. If the value in `ioVolIndex` is 0, the File Manager attempts to access the volume using only the value in this field. On output, the volume reference number.

`ioXVersion`

On input, the version of the extended volume parameter block. Currently, this value is 0.

`ioVolIndex`

On input, an index used for indexing through all the mounted volumes. If this value is positive, the File Manager uses it to find the volume for which to return information. For instance, if the value of `ioVolIndex` is 2, the File Manager attempts to access the second mounted volume in the VCB queue. If `ioVolIndex` is negative, the File Manager uses the values in the `ioNamePtr` and `ioVRefNum` fields to access the requested volume. If `ioVolIndex` is 0, the File Manager uses only the value in the `ioVRefNum` field.

`ioVCrDate`

On output, the date and time of the volume's creation (initialization).

`ioVLsMod`

On output, the date and time that the volume was last modified.

`ioVAtrb`

On output, the volume attributes. See "Volume Information Attribute Constants" (page 937) for a description of these attributes.

`ioVNmFls`

On output, the number of files in the root directory of the volume. For performance reasons, the Carbon File Manager does not return the number of files in this field; instead, it sets `ioVNmFls` to

0.To determine the number of files in the root directory of a volume in Carbon, call `PBGetCatInfoAsync` (page 648) for the root directory. The number of files in the root directory is returned in the `ioDrNmFls` field.

`ioVBitMap`
On output, the first block of the volume bitmap.

`ioVAllocPtr`
On output, the block where the next new file allocation search should start.

`ioVNmAlBlks`
On output, the number of allocation blocks on the volume.

`ioVAlBlkSiz`
On output, the allocation block size for the volume.

`ioVClpSiz`
On output, the volume's default clump size.

`ioAlBlSt`
On output, the first block in the volume block map.

`ioVNxtCNID`
On output, the next unused catalog node ID.

`ioVFrBlk`
On output, the number of free (unused) allocation blocks on the volume.

`ioVSigWord`
On output, the volume signature. For HFS volumes, this is 'BD' for HFS Plus volumes, this is 'H+'.

`ioVDrvInfo`
On output, the drive number. You can determine whether the given volume is online by inspecting the value of this field. For online volumes, the `ioVDrvInfo` field contains the drive number of the drive containing the specified volume and hence is always greater than 0. If the value returned in `ioVDrvInfo` is 0, the volume is either offline or ejected.

`ioVDRefNum`
On output, the driver reference number. You can determine whether the volume is offline or ejected by inspecting the value of this field. If the volume is offline, the value of `ioVDRefNum` is the negative of the drive number (which is cleared when the volume is placed offline; hence the `ioVDrvInfo` field for an offline volume is zero), and is a negative number. If the volume is ejected, the value of `ioVDRefNum` is the drive number itself, and thus is a positive number. For online volumes, `ioVDRefNum` contains a driver reference number; these numbers are always less than 0.

`ioVFSID`
On output, the file system ID for the file system handling this volume.

`ioVBkUp`
On output, the date and time that the volume was last backed up.

`ioVSeqNum`
Used internally.

`ioVWrCnt`
On output, the volume write count.

`ioVFilCnt`
On output, the number of files on the volume.

`ioVDirCnt`
On output, the number of directories on the volume.

`ioVFndrInfo`
On output, Finder information for the volume.

`ioVTotalBytes`
On output, the total number of bytes on the volume.

`ioVFreeBytes`
On output, the number of free bytes on the volume.

The `PBXGetVolInfoAsync` function is similar to the `PBHGetVInfoAsync` (page 686) function except that it returns additional volume space information in 64-bit integers and does not modify the information copied from the volume's volume control block (VCB). Systems that support `PBXGetVolInfoAsync` will have the `gestaltFSSupports2TBVols` bit set in the response returned by the `gestaltFSAttr Gestalt` selector. See *Inside Mac OS X: Gestalt Manager Reference* for a description of the `gestaltFSAttr` selector and of the bits that may be returned in the response.

Special Considerations

After an operation that changes the amount of free space on the volume—such as deleting a file—there may be a delay before a call to `PBXGetVolInfoAsync` returns the updated amount. This is because the File Manager caches and periodically updates file system information, to reduce the number of calls made to retrieve the information from the file system. Currently, the File Manager updates its information every 15 seconds. This primarily affects NFS volumes. DOS, SMB, UFS and WebDAV volumes were also affected by this in previous versions of Mac OS X, but behave correctly in Mac OS X version 10.3 and later.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBXGetVolInfoSync

Returns information about a volume, including size information for volumes up to 2 terabytes. (Deprecated in Mac OS X v10.4. Use `FSGetVolumeInfo` (page 500) instead.)

```
OSErr PBXGetVolInfoSync (
   XVolumeParamPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to an extended volume parameter block. See `XVolumeParam` (page 882) for a description of the `XVolumeParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a buffer. You should pass a pointer to a `Str31` value if you want the volume name returned; otherwise, pass `NULL`. If you specify a negative number in the `ioVolIndex` field, this

buffer should hold the name of the volume for which to return information. On output, a pointer to the volume's name.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume. If the value in the `ioVolIndex` field is negative, the File Manager uses the name in the `ioNamePtr` field, along with the value in the `ioVRefNum` field, to determine the volume. If the value in `ioVolIndex` is 0, the File Manager attempts to access the volume using only the value in this field. On output, the volume reference number.

`ioXVersion`
On input, the version of the extended volume parameter block. Currently, this value is 0.

`ioVolIndex`
On input, an index used for indexing through all the mounted volumes. If this value is positive, the File Manager uses it to find the volume for which to return information. For instance, if the value of `ioVolIndex` is 2, the File Manager attempts to access the second mounted volume in the VCB queue. If `ioVolIndex` is negative, the File Manager uses the values in the `ioNamePtr` and `ioVRefNum` fields to access the requested volume. If `ioVolIndex` is 0, the File Manager uses only the value in the `ioVRefNum` field.

`ioVCrDate`
On output, the date and time of the volume's creation (initialization).

`ioVLsMod`
On output, the date and time that the volume was last modified.

`ioVAtrb`
On output, the volume attributes. See "Volume Information Attribute Constants" (page 937) for a description of these attributes.

`ioVNmFls`
On output, the number of files in the root directory of the volume. For performance reasons, the Carbon File Manager does not return the number of files in this field; instead, it sets `ioVNmFls` to 0.To determine the number of files in the root directory of a volume in Carbon, call `PBGetCatInfoSync` (page 651) for the root directory. The number of files in the root directory is returned in the `ioDrNmFls` field.

`ioVBitMap`
On output, the first block of the volume bitmap.

`ioVAllocPtr`
On output, the block where the next new file allocation search should start.

`ioVNmAlBlks`
On output, the number of allocation blocks on the volume.

`ioVAlBlkSiz`
On output, the allocation block size for the volume.

`ioVClpSiz`
On output, the volume's default clump size.

`ioAlBlSt`
On output, the first block in the volume block map.

`ioVNxtCNID`
On output, the next unused catalog node ID.

`ioVFrBlk`
On output, the number of free (unused) allocation blocks on the volume.

`ioVSigWord`
: On output, the volume signature. For HFS volumes, this is 'BD' for HFS Plus volumes, this is 'H+'.

`ioVDrvInfo`
: On output, the drive number. You can determine whether the given volume is online by inspecting the value of this field. For online volumes, the `ioVDrvInfo` field contains the drive number of the drive containing the specified volume and hence is always greater than 0. If the value returned in `ioVDrvInfo` is 0, the volume is either offline or ejected.

`ioVDRefNum`
: On output, the driver reference number. You can determine whether the volume is offline or ejected by inspecting the value of this field. If the volume is offline, the value of `ioVDRefNum` is the negative of the drive number (which is cleared when the volume is placed offline; hence the `ioVDrvInfo` field for an offline volume is zero), and is a negative number. If the volume is ejected, the value of `ioVDRefNum` is the drive number itself, and thus is a positive number. For online volumes, `ioVDRefNum` contains a driver reference number; these numbers are always less than 0.

`ioVFSID`
: On output, the file system ID for the file system handling this volume.

`ioVBkUp`
: On output, the date and time that the volume was last backed up.

`ioVSeqNum`
: Used internally.

`ioVWrCnt`
: On output, the volume write count.

`ioVFilCnt`
: On output, the number of files on the volume.

`ioVDirCnt`
: On output, the number of directories on the volume.

`ioVFndrInfo`
: On output, Finder information for the volume.

`ioVTotalBytes`
: On output, the total number of bytes on the volume.

`ioVFreeBytes`
: On output, the number of free bytes on the volume.

The `PBXGetVolInfoSync` function is similar to the `PBHGetVInfoSync` (page 690) function except that it returns additional volume space information in 64-bit integers and does not modify the information copied from the volume's volume control block (VCB). Systems that support `PBXGetVolInfoSync` will have the `gestaltFSSupports2TBVols` bit set in the response returned by the `gestaltFSAttr Gestalt` selector. See *Inside Mac OS X: Gestalt Manager Reference* for a description of the `gestaltFSAttr` selector and of the bits that may be returned in the response.

Special Considerations

After an operation that changes the amount of free space on the volume—such as deleting a file—there may be a delay before a call to `PBXGetVolInfoSync` returns the updated amount. This is because the File Manager caches and periodically updates file system information, to reduce the number of calls made to retrieve the information from the file system. Currently, the File Manager updates its information every 15 seconds. This primarily affects NFS volumes. DOS, SMB, UFS and WebDAV volumes were also affected by this in previous versions of Mac OS X, but behave correctly in Mac OS X version 10.3 and later.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBXLockRangeAsync

Locks a range of bytes of the specified fork.

```
OSStatus PBXLockRangeAsync (
    FSRangeLockParamPtr paramBlock
);
```

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

Declared In
`Files.h`

PBXLockRangeSync

Locks a range of bytes of the specified fork.

```
OSStatus PBXLockRangeSync (
    FSRangeLockParamPtr paramBlock
);
```

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

Declared In
`Files.h`

PBXUnlockRangeAsync

Unlocks a range of bytes of the specified fork.

```
OSStatus PBXUnlockRangeAsync (
    FSRangeLockParamPtr paramBlock
);
```

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

Declared In
`Files.h`

PBXUnlockRangeSync

Unlocks a range of bytes of the specified fork.

```
OSStatus PBXUnlockRangeSync (
    FSRangeLockParamPtr paramBlock
);
```

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

Declared In
`Files.h`

SetEOF

Sets the logical size of an open file. (Deprecated in Mac OS X v10.4. Use `FSSetForkSize` (page 542) instead.)

```
OSErr SetEOF (
   FSIORefNum refNum,
   SInt32 logEOF
);
```

Parameters

`refNum`

The file reference number of an open file.

`logEOF`

The new logical size (the logical end-of-file) of the given file. If you set the `logEOF` parameter to 0, all space occupied by the file on the volume is released. The file still exists, but it contains 0 bytes. Setting a file fork's end-of-file to 0 is therefore not the same as deleting the file, which removes both file forks at once.

Return Value
A result code. See "File Manager Result Codes" (page 943).

Discussion
If you attempt to set the logical end-of-file beyond the physical end-of-file, the physical end-of-file is set 1 byte beyond the end of the next free allocation block if there isn't enough space on the volume, no change is made, and `SetEOF` returns `dskFulErr` as its function result.

To ensure that your changes to the file are written to disk, call one of the functions, `FlushVol` (page 466) , `PBFlushVolSync` (page 641) , or `PBFlushVolAsync` (page 640). To set the size of a named fork other than the data and resource forks, or to grow the size of a file beyond 2GB, you must use the `FSSetForkSize` (page 542) function, or one of the corresponding parameter block calls, `PBSetForkSizeSync` (page 762) and `PBSetForkSizeAsync` (page 761).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

SetFPos

Sets the position of the file mark. (Deprecated in Mac OS X v10.4. Use `FSSetForkPosition` (page 541) instead.)

```
OSErr SetFPos (
   FSIORefNum refNum,
   SInt16 posMode,
   SInt32 posOff
);
```

Parameters

refNum

The file reference number of an open file.

posMode

A constant specifying how to position the file mark; this parameter must contain one of the values described in "Position Mode Constants" (page 928).

posOff

The offset from the base location specified by the `posMode` parameter for the new file mark position. If you specify `fsFromLEOF` in the `posMode` parameter, the value in the `posOff` parameter must be less than or equal to 0. If you specify `fsAtMark`, the value in the `posOff` parameter is ignored.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

Because the read and write operations performed by the functions `FSRead` (page 536) and `FSWrite` (page 546) begin at the current mark, you may want to call `SetFPos` to reposition the file mark before reading from or writing to the file.

To set the file mark position for a named fork other than the data and resource forks, or to position the file mark at a point more than 2GB into the file, use the `FSSetForkPosition` (page 541) function, or one of the corresponding parameter block calls, `PBSetForkPositionSync` (page 760) and `PBSetForkPositionAsync` (page 759).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

UnmountVol

Unmounts a volume that isn't currently being used. (Deprecated in Mac OS X v10.4. Use `FSUnmountVolumeSync` (page 545) instead.)

```
OSErr UnmountVol (
   ConstStr63Param volName,
   FSVolumeRefNum vRefNum
);
```

Parameters

volName

The name of a mounted volume. This parameter may be `NULL`.

vRefNum

The volume reference number, drive number, or 0 for the default volume.

Return Value

A result code. See "File Manager Result Codes" (page 943).

Discussion

All files on the volume (except those opened by the Operating System) must be closed before you call `UnmountVol`, which does not eject the volume.

Most applications do not need to use this function, because the user typically ejects (and possibly also unmounts) a volume in the Finder.

Special Considerations

Don't unmount the startup volume. Doing so will cause a system crash.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

Callbacks by Task

File Operation Callbacks

`FSFileOperationStatusProcPtr` (page 790)

Defines a status callback function for an asynchronous file operation on an `FSRef` object.

`FSPathFileOperationStatusProcPtr` (page 791)

Defines a status callback function for an asynchronous file operation on an object specified with a pathname.

Miscellaneous Callbacks

`FNSubscriptionProcPtr` (page 789)

Callback delivered for directory notifications.

`FSVolumeEjectProcPtr` (page 792)

FSVolumeMountProcPtr (page 792)

FSVolumeUnmountProcPtr (page 793)

IOCompletionProcPtr (page 794)
Defines a pointer to a completion function. Your completion function is executed by the File Manager after the completion of an asynchronous File Manager function call.

Callbacks

FNSubscriptionProcPtr

Callback delivered for directory notifications.

```
typedef void (*FNSubscriptionProcPtr) (
    FNMessage message,
    OptionBits flags,
    void * refcon,
    FNSubscriptionRef subscription
);
```

If you name your function `MyFNSubscriptionProc`, you would declare it like this:

```
void MyFNSubscriptionProc (
    FNMessage message,
    OptionBits flags,
    void * refcon,
    FNSubscriptionRef subscription
);
```

Parameters

message
An indication of what happened.

flags
Options regarding the delivery of the notification; typically `kNilOptions`.

refcon
A pointer to a user reference supplied with subscription.

subscription
A subscription corresponding to this notification.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`Files.h`

FSFileOperationStatusProcPtr

Defines a status callback function for an asynchronous file operation on an `FSRef` object.

```
typedef void (*FSFileOperationStatusProcPtr) (
    FSFileOperationRef fileOp,
    const FSRef *currentItem,
    FSFileOperationStage stage,
    OSStatus error,
    CFDictionaryRef statusDictionary,
    void *info
);
```

If you name your function `MyFSFileOperationStatusProc`, you would declare it like this:

```
void MyFSFileOperationStatusProc (
    FSFileOperationRef fileOp,
    const FSRef *currentItem,
    FSFileOperationStage stage,
    OSStatus error,
    CFDictionaryRef statusDictionary,
    void *info
);
```

Parameters

fileOp

The file operation.

currentItem

A pointer to an `FSRef` variable. On output, the variable contains the object currently being moved or copied. If the operation is complete, this parameter refers to the target (the new object corresponding to the source object in the destination directory).

stage

The current stage of the operation.

error

The current error status of the operation.

statusDictionary

A dictionary with more detailed status information. For information about the contents of the dictionary, see "File Operation Status Dictionary Keys" (page 919). You are not responsible for releasing the dictionary.

info

A pointer to user-defined data associated with this operation.

Discussion

When you call `FSCopyObjectAsync` (page 477), `FSMoveObjectAsync` (page 511), or `FSMoveObjectToTrashAsync` (page 513), you can specify a status callback function of this type. The function you provide is called by the File Manager whenever the file operation changes stages (including failing due to an error), or as updated information is available limited by the status change interval of the operation. If you need to save any of the status information beyond the scope of the callback, you should make a copy of the information.

Availability

Available in Mac OS X v10.4 and later.

Declared In
`Files.h`

FSPathFileOperationStatusProcPtr

Defines a status callback function for an asynchronous file operation on an object specified with a pathname.

```
typedef void (*FSPathFileOperationStatusProcPtr) (
    FSFileOperationRef fileOp,
    const char *currentItem,
    FSFileOperationStage stage,
    OSStatus error,
    CFDictionaryRef statusDictionary,
    void *info
);
```

If you name your function `MyFSPathFileOperationStatusProc`, you would declare it like this:

```
void MyFSPathFileOperationStatusProc (
    FSFileOperationRef fileOp,
    const char *currentItem,
    FSFileOperationStage stage,
    OSStatus error,
    CFDictionaryRef statusDictionary,
    void *info
);
```

Parameters

fileOp
The file operation.

currentItem
The UTF-8 pathname of the object currently being moved or copied. If the operation is complete, this parameter refers to the target (the new object corresponding to the source object in the destination directory).

stage
The current stage of the operation.

error
The current error status of the operation.

statusDictionary
A dictionary with more detailed status information. For information about the contents of the dictionary, see "File Operation Status Dictionary Keys" (page 919). You are not responsible for releasing the dictionary.

info
A pointer to user-defined data associated with this operation.

Discussion

When you call `FSPathCopyObjectAsync` (page 517), `FSPathMoveObjectAsync` (page 521), or `FSPathMoveObjectToTrashAsync` (page 523), you can specify a status callback function of this type. The function you provide is called by the File Manager whenever the file operation changes stages (including failing due to an error), or as updated information is available limited by the status change interval of the operation. If you need to save any of the status information beyond the scope of the callback, you should make a copy of the information.

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

Declared In
`Files.h`

FSVolumeEjectProcPtr

```
typedef void (*FSVolumeEjectProcPtr) (
    FSVolumeOperation volumeOp,
    void * clientData,
    OSStatus err,
    FSVolumeRefNum volumeRefNum,
    pid_t dissenter
);
```

If you name your function `MyFSVolumeEjectProc`, you would declare it like this:

```
void MyFSVolumeEjectProc (
    FSVolumeOperation volumeOp,
    void * clientData,
    OSStatus err,
    FSVolumeRefNum volumeRefNum,
    pid_t dissenter
);
```

Parameters

volumeOp

clientData

err

volumeRefNum

dissenter

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

Declared In
`Files.h`

FSVolumeMountProcPtr

```
typedef void (*FSVolumeMountProcPtr) (
    FSVolumeOperation volumeOp,
    void * clientData,
    OSStatus err,
    FSVolumeRefNum mountedVolumeRefNum
);
```

If you name your function `MyFSVolumeMountProc`, you would declare it like this:

```
void MyFSVolumeMountProc (
    FSVolumeOperation volumeOp,
```

```
    void * clientData,
    OSStatus err,
    FSVolumeRefNum mountedVolumeRefNum
);
```

Parameters

volumeOp

clientData

err

mountedVolumeRefNum

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

Declared In
`Files.h`

FSVolumeUnmountProcPtr

```
typedef void (*FSVolumeUnmountProcPtr) (
    FSVolumeOperation volumeOp,
    void * clientData,
    OSStatus err,
    FSVolumeRefNum volumeRefNum,
    pid_t dissenter
);
```

If you name your function `MyFSVolumeUnmountProc`, you would declare it like this:

```
void MyFSVolumeUnmountProc (
    FSVolumeOperation volumeOp,
    void * clientData,
    OSStatus err,
    FSVolumeRefNum volumeRefNum,
    pid_t dissenter
);
```

Parameters

volumeOp

clientData

err

volumeRefNum

dissenter

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

Declared In
`Files.h`

IOCompletionProcPtr

Defines a pointer to a completion function. Your completion function is executed by the File Manager after the completion of an asynchronous File Manager function call.

```
typedef void (*IOCompletionProcPtr) (
    ParmBlkPtr paramBlock
);
```

If you name your function `MyIOCompletionProc`, you would declare it like this:

```
void MyIOCompletionProc (
    ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the parameter block that was passed to the asynchronous File Manager function.

Return Value

Discussion

When you execute an asynchronous File Manager function (an `Async` function), you can specify a completion routine by passing the routine's address in the `ioCompletion` field of the parameter block passed to the function. Because you requested asynchronous execution, the File Manager places an I/O request in the file I/O queue and returns control to your application—possibly even before the actual I/O operation is completed. The File Manager takes requests from the queue one at a time and processes them meanwhile, your application is free to do other processing.

A function executed asynchronously returns control to your application with the result code `noErr` as soon as the call is placed in the file I/O queue. This result code does not indicate that the call has successfully completed, but simply indicates that the call was successfully placed in the queue. To determine when the call is actually completed, you can inspect the `ioResult` field of the parameter block. This field is set to a positive number when the call is made and set to the actual result code when the call is completed. If you specify a completion routine, it is executed after the result code is placed in `ioResult`.

The File Manager, when the File Sharing or AppleShare file server is active, will execute requests in arbitrary order. That means that if there is a request that depends on the completion of a previous request, it is an error for your program to issue the second request until the completion of the first request. For example, issuing a write request and then issuing a read request for the same data isn't guaranteed to read back what was written unless the read request isn't made until after the write request completes.

Request order can also change if a call results in a disk switch dialog to bring an offline volume back online.

Special Considerations

Because a completion routine is executed at interrupt time, it should not allocate, move, or purge memory (either directly or indirectly) and should not depend on the validity of handles to unlocked blocks.

If your completion routine uses application global variables, it must also ensure that register A5 contains the address of the boundary between your application global variables and your application parameters.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

Data Types

AccessParam

Defines a parameter block used by low-level HFS file and directory access rights manipulation functions.

```
struct AccessParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short filler3;
    short ioDenyModes;
    short filler4;
    SInt8 filler5;
    SInt8 ioACUser;
    long filler6;
    long ioACOwnerID;
    long ioACGroupID;
    long ioACAccess;
    long ioDirID;
};
typedef struct AccessParam AccessParam;
typedef AccessParam * AccessParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`filler3`
: Reserved.

`ioDenyModes`
: Access mode information.

`filler4`
: Reserved.

`filler5`
: Reserved.

`ioACUser`
: The user's access rights for the specified directory.

`filler6`
: Reserved.

`ioACOwnerID`
: The owner ID.

`ioACGroupID`
: The group ID.

`ioACAccess`
: The directory access privileges.

`ioDirID`

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

AFPAlternateAddress

Defines a block of tagged addresses for AppleShare clients.

```
struct AFPAlternateAddress {
    UInt8 fVersion;
    UInt8 fAddressCount;
    UInt8 fAddressList[1];
};
typedef struct AFPAlternateAddress AFPAlternateAddress;
```

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

Declared In
`Files.h`

AFPTagData

Defines a structure which contains tagged address information for AppleShare clients.

```
struct AFPTagData {
    UInt8 fLength;
    UInt8 fType;
    UInt8 fData[1];
};
typedef struct AFPTagData AFPTagData;
```

Fields

`fLength`

The length, in bytes, of this data tag, including the `fLength` field itself. See "AFP Tag Length Constants" (page 885).

`fType`

The type of the data tag. See "AFP Tag Type Constants" (page 886) for the constants which you can use here.

`fData`

Variable length data, containing the address.

Discussion

The new tagged data format for addressing allows for changes in addressing formats, allowing AppleShare clients to support new addressing standards without changing the interface. The `AFPAlternateAddress` (page 796) data structure uses the `AFPTagData` structure to specify a tagged address.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

AFPVolMountInfo

Defines a volume mounting structure for an AppleShare server.

```
struct AFPVolMountInfo {
    short length;
    VolumeType media;
    short flags;
    SInt8 nbpInterval;
    SInt8 nbpCount;
    short uamType;
    short zoneNameOffset;
    short serverNameOffset;
    short volNameOffset;
    short userNameOffset;
    short userPasswordOffset;
    short volPasswordOffset;
    char AFPData[144];
};
typedef struct AFPVolMountInfo AFPVolMountInfo;
typedef AFPVolMountInfo * AFPVolMountInfoPtr;
```

Fields

`length`
: The length of the `AFPVolMountInfo` structure (that is, the total length of the structure header described here plus the variable-length location data).

`media`
: The volume type of the remote volume. The value `AppleShareMediaType` (a constant that translates to `'afpm'`) represents an AppleShare volume.

`flags`
: If bit 0 is set, no greeting message from the server is displayed.

`nbpInterval`
: The NBP retransmit interval, in units of 8 ticks.

`nbpCount`
: The NBP retransmit count. This field specifies the total number of times a packet should be transmitted, including the first transmission.

`uamType`
: The user authentication method used by the remote volume. AppleShare uses four methods, defined by the constants described in "Authentication Method Constants" (page 888).

`zoneNameOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the name (as a pascal string) of the AppleShare zone.

`serverNameOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the name (as a pascal string) of the AppleShare server.

`volNameOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the name (as a pascal string) of the volume.

`userNameOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the name (as a pascal string) of the user.

`userPasswordOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the user's password (as a pascal string).

`volPasswordOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the volume's password (as a pascal string). Some versions of the AppleShare software do not pass the information in this field to the server.

`AFPData`
: The actual volume mounting information, offsets to which are contained in the preceding six fields. To mount an AFP volume, you must fill in the structure with at least the zone name, server name, user name, user password, and volume password. You can lay out the data in any order within this data field, as long as you specify the correct offsets in the offset fields.

Discussion
The only volumes that currently support the programmatic mounting functions are AppleShare servers, which use a volume mounting structure of type `AFPVolMountInfo`.

To mount an AppleShare server, fill out an `AFPVolMountInfo` structure using the `PBGetVolMountInfo` function and then pass this structure to the `PBVolumeMount` function to mount the volume.

Version Notes
AppleShare clients prior to version 3.7 mount volumes over AppleTalk only. For maximum compatibility set the `uamType` field to 1 for guest login or 3 for login using a password.

To mount volumes using IP addresses and other address formats, use the `AFPXVolMountInfo` (page 799) structure.

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

Declared In
`Files.h`

AFPXVolMountInfo

Defines a volume mounting structure for an AppleShare server, for AppleShare 3.7 and later.

```
struct AFPXVolMountInfo {
    short length;
    VolumeType media;
    short flags;
    SInt8 nbpInterval;
    SInt8 nbpCount;
    short uamType;
    short zoneNameOffset;
    short serverNameOffset;
    short volNameOffset;
    short userNameOffset;
    short userPasswordOffset;
    short volPasswordOffset;
    short extendedFlags;
    short uamNameOffset;
    short alternateAddressOffset;
    char AFPData[176];
};
typedef struct AFPXVolMountInfo AFPXVolMountInfo;
typedef AFPXVolMountInfo * AFPXVolMountInfoPtr;
```

Fields

`length`
: The length of the `AFPXVolMountInfo` structure (that is, the total length of the structure header described here plus the variable-length location data).

`media`
: The volume type of the remote volume. The value `AppleShareMediaType` (a constant that translates to `'afpm'`) represents an AppleShare volume.

`flags`
: Volume mount flags. See "Volume Mount Flags" (page 942) for a description of the bits in this field. In order to use the new features of the extended AFP volume mount structure, you must set the `volMountExtendedFlagsBit` bit.

`nbpInterval`
: The NBP retransmit interval, in units of 8 ticks.

`nbpCount`
: The NBP retransmit count. This field specifies the total number of times a packet should be transmitted, including the first transmission.

`uamType`
: The user authentication method used by the remote volume. AppleShare uses four methods, defined by the constants described in "Authentication Method Constants" (page 888).

`zoneNameOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the name (as a pascal string) of the AppleShare zone.

`serverNameOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the name (as a pascal string) of the AppleShare server.

`volNameOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the name (as a pascal string) of the volume.

`userNameOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the name (as a pascal string) of the user.

`userPasswordOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the user's password (as a pascal string).

`volPasswordOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the volume's password (as a pascal string). Some versions of the AppleShare software do not pass the information in this field to the server.

`extendedFlags`
: Extended flags. See "Extended AFP Volume Mounting Information Flag" (page 903).

`uamNameOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the user authentication module name (as a pascal string).

`alternateAddressOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing IP addresses, specified as a block of tagged data. This block of tagged data begins with a version byte and a count byte, followed by up to 255 tagged addresses. See `AFPAlternateAddress` (page 796).

`AFPData`
: The actual volume mounting information, offsets to which are contained in the preceding fields. To mount an AFP volume, you must fill in the structure with at least the zone name, server name, user name, user password, and volume password. You can lay out the data in any order within this data field, as long as you specify the correct offsets in the offset fields.

Discussion
To mount an AppleShare server, fill out an `AFPXVolMountInfo` structure using the `PBGetVolMountInfo` function and then pass this structure to the `PBVolumeMount` function to mount the volume.

The extended AFP volume mount information structure requires AppleShare client 3.7 and later. The new fields and flag bits allow you to specify the information needed to support TCP/IP and User Authentication Modules.

Note that, for all fields specifying an offset, if you wish to leave the string field in the `AFPData` field empty, you must specify an empty string and have the offset in the corresponding offset field point to that empty string. You cannot simply pass 0 as the offset.

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

Declared In
`Files.h`

CatPositionRec

Defines a catalog position structure, which maintains the current position of a catalog search between calls to `PBCatSearchSync` or `PBCatSearchAsync`.

```
struct CatPositionRec {
    long initialize;
    short priv[6];
};
typedef struct CatPositionRec CatPositionRec;
```

Fields

`initialize`

The starting point of the catalog search. To start searching at the beginning of a catalog, specify 0 in this field. To resume a previous search, pass the value returned by the previous call to `PBCatSearchSync` or `PBCatSearchAsync`.

`priv`

An array of integers that is used internally by `PBCatSearchSync` and `PBCatSearchAsync`.

Discussion

When you call the `PBCatSearchSync` or `PBCatSearchAsync` function to search a volume's catalog file, you can specify, in the `ioCatPosition` field of the parameter block passed to `PBCatSearchSync` and `PBCatSearchAsync`, a catalog position structure. If a catalog search consumes more time than is allowed by the `ioSearchTime` field, `PBCatSearchSync` and `PBCatSearchAsync` store a directory-location index in that structure; when you call `PBCatSearchSync` or `PBCatSearchAsync` again, it uses that structure to resume searching where it left off.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

CInfoPBRec

Defines a catalog information parameter block for file and directory information.

```
union CInfoPBRec {
    HFileInfo hFileInfo;
    DirInfo dirInfo;
};
typedef union CInfoPBRec CInfoPBRec;
typedef CInfoPBRec * CInfoPBPtr;
```

Fields

`hFileInfo`

`dirInfo`

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Files.h`

CMovePBRec

Defines a parameter block, used with the functions `PBCatMoveSync` and `PBCatMoveAsync`.

```
struct CMovePBRec {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    long filler1;
    StringPtr ioNewName;
    long filler2;
    long ioNewDirID;
    long filler3[2];
    long ioDirID;
};
typedef struct CMovePBRec CMovePBRec;
typedef CMovePBRec * CMovePBPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type (This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`filler1`
Reserved.

`ioNewName`
The name of the directory into which the specified file or directory is to be moved.

`filler2`
Reserved.

`ioNewDirID`
: The directory ID of the directory into which the specified file or directory is to be moved.

`filler3`
: Reserved.

`ioDirID`
: The current directory ID of the file or directory to be moved (used in conjunction with the `ioVRefNum` and `ioNamePtr` fields).

Discussion
The low-level HFS function `PBCatMove` uses the catalog move parameter block defined by the `CMovePBRec` data type.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

CntrlParam

Defines a parameter block used by control and status functions in the classic Device Manager.

```
struct CntrlParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioCRefNum;
    short csCode;
    short csParam[11];
};
typedef struct CntrlParam CntrlParam;
typedef CntrlParam * CntrlParamPtr;
```

Fields
`qLink`
: A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
: The queue type. This field is used internally by the File Manager.

`ioTrap`
: The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
: The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
: A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`ioCRefNum`
: The driver reference number for the I/O operation.

`csCode`
: A value identifying the type of control or status request. Each driver may interpret this number differently.

`csParam`
: The control or status information passed to or from the driver. This field is declared generically as an array of eleven integers. Each driver may interpret the contents of this field differently. Refer to the driver's documentation for specific information.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

ConstFSSpecPtr

Defines a pointer to an `FSSpec` structure.

```
typedef const FSSpec*  ConstFSSpecPtr;
```

Discussion
The only difference between "`const FSSpec*`" and the `ConstFSSpecPtr` data type is that, as a parameter, a `ConstFSSpecPtr` data type is allowed to be `NULL`. See `FSSpec` (page 840).

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

Declared In
`Files.h`

ConstHFSUniStr255Param

Defines a pointer to an `HFSUniStr255` structure.

```
typedef const HFSUniStr255*  ConstHFSUniStr255Param;
```

Discussion
See `HFSUniStr255` (page 855).

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

Declared In
`Files.h`

CopyParam

Defines a parameter block used by low-level HFS file copying functions.

```
struct CopyParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioDstVRefNum;
    short filler8;
    StringPtr ioNewName;
    StringPtr ioCopyName;
    long ioNewDirID;
    long filler14;
    long filler15;
    long ioDirID;
};
typedef struct CopyParam CopyParam;
typedef CopyParam * CopyParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
: A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`ioDstVRefNum`
: A volume reference number for the destination volume.

`filler8`
: Reserved.

`ioNewName`
: A pointer to the destination pathname.

`ioCopyName`
: A pointer to an optional name.

`ioNewDirID`
: A destination directory ID.

`filler14`
: Reserved.

`filler15`
: Reserved.

`ioDirID`
: A directory ID.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

CSParam

Defines a parameter block used by low-level HFS catalog search functions.

```
struct CSParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    FSSpecPtr ioMatchPtr;
    long ioReqMatchCount;
    long ioActMatchCount;
    long ioSearchBits;
    CInfoPBPtr ioSearchInfo1;
    CInfoPBPtr ioSearchInfo2;
    long ioSearchTime;
    CatPositionRec ioCatPosition;
    Ptr ioOptBuffer;
    long ioOptBufSize;
};
typedef struct CSParam CSParam;
typedef CSParam * CSParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`ioMatchPtr`

A pointer to an array of `FSSpec` (page 840) structures in which the file and directory names that match the selection criteria are returned. The array must be large enough to hold the largest possible number of `FSSpec` structures, as determined by the `ioReqMatchCount` field.

`ioReqMatchCount`

The maximum number of matches to return. This number should be the number of `FSSpec` structures that will fit in the memory pointed to by the `ioMatchPtr` field. You can use this field to avoid a possible excess of matches for criteria that prove to be too general (or to limit the length of a search if the `ioSearchTime` field isn't used).

`ioActMatchCount`

The number of actual matches found.

`ioSearchBits`

The fields of the parameter blocks in the `ioSearchInfo1` and `ioSearchInfo2` fields that are relevant to the search. See "Catalog Search Bits" (page 896) for more information.

`ioSearchInfo1`

A pointer to a `CInfoPBRec` parameter block that contains the search information. For values that match by mask and value (Finder information, for example), set the bits in the structure passed in the `ioSearchInfo2` field, and set the matching values in this structure. For values that match against a range (such as dates), set the lower bounds for the range in this structure.

`ioSearchInfo2`

A pointer to a second `CInfoPBRec` parameter block that contains the search information. For values that match by mask and value (Finder information, for example), set the bits in this structure, and set the matching values in the structure passed in the `ioSearchInfo1` field. For values that match against a range (such as dates), set the upper bounds for the range in this structure.

`ioSearchTime`

A time limit on a search, in Time Manager format. Use this field to limit the run time of a single call to `PBCatSearchSync` or `PBCatSearchAsync`. A value of 0 imposes no time limit. If the value of this field is positive, it is interpreted as milliseconds. If the value of this field is negative, it is interpreted as negated microseconds.

`ioCatPosition`

A position in the catalog where searching should begin. Use this field to keep an index into the catalog when breaking down the `PBCatSearchSync` or `PBCatSearchAsync` search into a number of smaller searches. This field is valid whenever `PBCatSearchSync` or `PBCatSearchAsync` exits because it either spends the maximum time allowed by `ioSearchTime` or finds the maximum number of matches allowed by `ioReqMatchCount`.

To start at the beginning of the catalog, set the `initialize` field of `ioCatPosition` to 0. Before exiting after an interrupted search, `PBCatSearchSync` or `PBCatSearchAsync` sets that field to the next catalog entry to be searched.

To resume where the previous call stopped, pass the entire `CatPositionRec` (page 801) structure returned by the previous call as input to the next.

`ioOptBuffer`

A pointer to an optional read buffer. The `ioOptBuffer` and `ioOptBufSize` fields let you specify a part of memory as a read buffer, increasing search speed.

`ioOptBufSize`

The size of the buffer pointed to by `ioOptBuffer`. Buffer size effectiveness varies with models and configurations, but a 16 KB buffer is likely to be optimal. The size should be at least 1024 bytes and should be an integral multiple of 512 bytes.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

DirInfo

Defines a structure which holds catalog information about a directory.

```
struct DirInfo {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioFRefNum;
    SInt8 ioFVersNum;
    SInt8 filler1;
    short ioFDirIndex;
    SInt8 ioFlAttrib;
    SInt8 ioACUser;
    DInfo ioDrUsrWds;
    long ioDrDirID;
    unsigned short ioDrNmFls;
    short filler3[9];
    unsigned long ioDrCrDat;
    unsigned long ioDrMdDat;
    unsigned long ioDrBkDat;
    DXInfo ioDrFndrInfo;
    long ioDrParID;
};
typedef struct DirInfo DirInfo;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`ioFRefNum`
: The file reference number of an open file.

`ioFVersNum`
: A file version number. This field is no longer used. File version numbers are an artifact of the obsolete MFS, and are not supported on HFS volumes. You should always set this field to 0.

`filler1`
: Reserved.

`ioFDirIndex`
: A file and directory index. If this field contains a positive number, `PBGetCatInfoSync` and `PBGetCatInfoAsync` return information about the file or directory having that directory index in the directory specified by the `ioVRefNum` field. (If `ioVRefNum` contains a volume reference number, the specified directory is that volume's root directory.)

  If this field contains 0, `PBGetCatInfoSync` and `PBGetCatInfoAsync` return information about the file or directory whose name is specified in the `ioNamePtr` field and that is located in the directory specified by the `ioVRefNum` field. (Once again, if `ioVRefNum` contains a volume reference number, the specified directory is that volume's root directory.)

  If this field contains a negative number, `PBGetCatInfoSync` and `PBGetCatInfoAsync` ignore the `ioNamePtr` field and returns information about the directory specified in the `ioDirID` field. If both `ioDirID` and `ioVRefNum` are set to 0, `PBGetCatInfoSync` and `PBGetCatInfoAsync` return information about the current default directory.

`ioFlAttrib`
: File or directory attributes. See "File Attribute Constants" (page 914) for the meaning of the bits in this field.

`ioACUser`
: The user's access rights for the specified directory. See "User Privileges Constants" (page 930) for the meaning of the bits in this field.

`ioDrUsrWds`
: Information used by the Finder.

`ioDrDirID`
: A directory ID. On input to `PBGetCatInfoSync` and `PBGetCatInfoAsync` , this field contains a directory ID, which is used only if the value of the `ioFDirIndex` field is negative. On output, this field contains the directory ID of the specified directory.

`ioDrNmFls`
: The number of files in the directory.

`filler3`
: Reserved.

`ioDrCrDat`

The date and time of the directory's creation, in seconds since midnight, January 1, 1904. However, on Mac OS X, if you set the creation date to a date between January 1, 1904 and January 1, 1970, it will be clipped to January 1, 1970, and that is the value which will be returned if you later try to retrieve the creation date.

Note that file systems other than AFP, HFS and HFS Plus do not generally support creation dates.

`ioDrMdDat`

The date and time of the last modification to the directory, in seconds since midnight, January 1, 1904. However, on Mac OS X, if you set the modification date to a date between January 1, 1904 and January 1, 1970, it will be clipped to January 1, 1970.

`ioDrBkDat`

The date and time that the directory was last backed up, in seconds since midnight, January 1, 1904. However, on Mac OS X, if you set the backup date to a date between January 1, 1904 and January 1, 1970, it will be clipped to January 1, 1970.

Note that file systems other than AFP, HFS and HFS Plus do not generally support backup dates.

`ioDrFndrInfo`

Additional information used by the Finder.

`ioDrParID`

The directory ID of the specified directory's parent directory.

`refCon`

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

DrvQEl

Defines a drive queue element.

```
struct DrvQEl {
    QElemPtr qLink;
    short qType;
    short dQDrive;
    short dQRefNum;
    short dQFSID;
    unsigned short dQDrvSz;
    unsigned short dQDrvSz2;
};
typedef struct DrvQEl DrvQEl;
typedef DrvQEl * DrvQElPtr;
```

Fields
`qLink`

A pointer to the next entry in the drive queue.

`qType`
: Used to specify the size of the drive. If the value of this field is 0, the number of logical blocks on the drive is contained in the `dQDrvSz` field alone. If the value of this field is 1, both the `dQDrvSz` field and the `dQDrvSz2` field are used to store the number of blocks; in that case, the `dQDrvSz2` field contains the high-order word of this number and `dQDrvSz` contains the low-order word.

`dQDrive`
: The drive number of the drive.

`dQRefNum`
: The driver reference number of the driver controlling the device on which the volume is mounted.

`dQFSID`
: An identifier for the file system handling the volume in the drive it's zero for volumes handled by the File Manager and nonzero for volumes handled by other file systems.

`dQDrvSz`
: The number of logical blocks on the drive.

`dQDrvSz2`
: An additional field to handle large drives. This field is only used if the `qType` field is 1.

Discussion
The File Manager maintains a list of all disk drives connected to the computer. It maintains this list in the drive queue, which is a standard operating system queue. The drive queue is initially created at system startup time. Elements are added to the queue at system startup time or when you call the `AddDrive` function. The drive queue can support any number of drives, limited only by memory space. Each element in the drive queue contains information about the corresponding drive.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

DTPBRec

Defines the desktop database parameter block used by the desktop database functions.

```
struct DTPBRec {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioDTRefNum;
    short ioIndex;
    long ioTagInfo;
    Ptr ioDTBuffer;
    long ioDTReqCount;
    long ioDTActCount;
    SInt8 ioFiller1;
    UInt8 ioIconType;
    short ioFiller2;
    long ioDirID;
    OSType ioFileCreator;
    OSType ioFileType;
    long ioFiller3;
    long ioDTLgLen;
    long ioDTPyLen;
    short ioFiller4[14];
    long ioAPPLParID;
};
typedef struct DTPBRec DTPBRec;
typedef DTPBRec * DTPBPtr;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`

The queue type. This field is used internally by the File Manager.

`ioTrap`

The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`

The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`

A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`

The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a file, directory, or volume name. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: The volume reference number.

`ioDTRefNum`
: The desktop database reference number.

`ioIndex`
: The index into icon list.

`ioTagInfo`
: The tag information.

`ioDTBuffer`
: The data buffer.

`ioDTReqCount`
: The requested length of data.

`ioDTActCount`
: The actual length of data.

`ioFiller1`
: Unused.

`ioIconType`
: The icon type.

`ioFiller2`
: Unused.

`ioDirID`
: The parent directory ID.

`ioFileCreator`
: The file creator.

`ioFileType`
: The file type.

`ioFiller3`
: Unused.

`ioDTLgLen`
: The logical length of the desktop database.

`ioDTPyLen`
: The physical length of the desktop database.

`ioFiller4`
: Unused.

`ioAPPLParID`
: The parent directory ID of an application.

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

Not available to 64-bit applications.

Declared In
`Files.h`

FCBPBRec

Defines a file control block (FCB) parameter block used by the functions `PBGetFCBInfoSync` and `PBGetFCBInfoAsync`.

```
struct FCBPBRec {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioRefNum;
    short filler;
    short ioFCBIndx;
    short filler1;
    long ioFCBFlNm;
    short ioFCBFlags;
    unsigned short ioFCBStBlk;
    long ioFCBEOF;
    long ioFCBPLen;
    long ioFCBCrPs;
    short ioFCBVRefNum;
    long ioFCBClpSiz;
    long ioFCBParID;
};
typedef struct FCBPBRec FCBPBRec;
typedef FCBPBRec * FCBPBPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`ioRefNum`
: The file reference number of an open file.

`filler`
: Reserved.

`ioFCBIndx`
: An index for use with the `PBGetFCBInfoSync` and `PBGetFCBInfoAsync` functions.

`filler1`
: Reserved.

`ioFCBFlNm`
: The file ID.

`ioFCBFlags`
: Flags describing the status of the file. See "FCB Flags" (page 906) for the meanings of the bits in this field.

`ioFCBStBlk`
: The number of the first allocation block of the file.

`ioFCBEOF`
: The logical length (logical end-of-file) of the file.

`ioFCBPLen`
: The physical length (physical end-of-file) of the file.

`ioFCBCrPs`
: The current position of the file mark.

`ioFCBVRefNum`
: The volume reference number.

`ioFCBClpSiz`
: The file clump size.

`ioFCBParID`
: The file's parent directory ID.

Discussion
The low-level HFS function `PBGetFCBInfo` uses the file control block parameter block defined by the `FCBPBRec` data type.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

FIDParam

Defines a parameter block used by low-level HFS file ID functions.

```
struct FIDParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    long filler14;
    StringPtr ioDestNamePtr;
    long filler15;
    long ioDestDirID;
    long filler16;
    long filler17;
    long ioSrcDirID;
    short filler18;
    long ioFileID;
};
typedef struct FIDParam FIDParam;
typedef FIDParam * FIDParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`filler14`
: Reserved.

`ioDestNamePtr`
: A pointer to the name of the destination file.

`filler15`
: Reserved.

`ioDestDirID`
: The parent directory ID of the destination file.

`filler16`
: Reserved.

`filler17`
: Reserved.

`ioSrcDirID`
: The parent directory ID of the source file.

`filler18`
: Reserved.

`ioFileID`
: The file ID.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

FileParam

Defines a parameter block used by low-level functions for getting and setting file information.

```
struct FileParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioFRefNum;
    SInt8 ioFVersNum;
    SInt8 filler1;
    short ioFDirIndex;
    SInt8 ioFlAttrib;
    SInt8 ioFlVersNum;
    FInfo ioFlFndrInfo;
    unsigned long ioFlNum;
    unsigned short ioFlStBlk;
    long ioFlLgLen;
    long ioFlPyLen;
    unsigned short ioFlRStBlk;
    long ioFlRLgLen;
    long ioFlRPyLen;
    unsigned long ioFlCrDat;
    unsigned long ioFlMdDat;
};
typedef struct FileParam FileParam;
typedef FileParam * FileParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`ioFRefNum`
The file reference number of an open file.

`ioFVersNum`
A file version number. This field is no longer used. File version numbers are an artifact of the obsolete MFS, and are not supported on HFS volumes. You should always set this field to 0.

`filler1`
Reserved.

`ioFDirIndex`
A directory index for use with the `PBHGetFInfoSync` (page 683) and `PBHGetFInfoAsync` (page 682) functions.

`ioFlAttrib`
File attributes. See "File Attribute Constants" (page 914) for the meaning of the bits in this field.

`ioFlVersNum`
A file version number. This feature is no longer supported, and you must always set this field to 0.

`ioFlFndrInfo`
Information used by the Finder.

`ioFlNum`
A file ID.

`ioFlStBlk`
The first allocation block of the data fork. This field contains 0 if the file's data fork is empty.

`ioFlLgLen`
The logical length (logical end-of-file) of the data fork.

`ioFlPyLen`
The physical length (physical end-of-file) of the data fork.

`ioFlRStBlk`
The first allocation block of the resource fork. This field contains 0 if the file's resource fork is empty.

`ioFlRLgLen`
The logical length (logical end-of-file) of the resource fork.

`ioFlRPyLen`
The physical length (physical end-of-file) of the resource fork.

`ioFlCrDat`
The date and time of the file's creation, specified in seconds since midnight, January 1, 1904.

`ioFlMdDat`
The date and time of the last modification to the file, specified in seconds since midnight, January 1, 1904.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

FNSubscriptionRef

```
typedef struct OpaqueFNSubscriptionRef * FNSubscriptionRef;
```

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

Declared In
`Files.h`

FNSubscriptionUPP

```
typedef FNSubscriptionProcPtr FNSubscriptionUPP;
```

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

Declared In
`Files.h`

ForeignPrivParam

Defines a parameter block used by low-level HFS foreign privileges functions.

```
struct ForeignPrivParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    long ioFiller21;
    long ioFiller22;
    Ptr ioForeignPrivBuffer;
    long ioForeignPrivActCount;
    long ioForeignPrivReqCount;
    long ioFiller23;
    long ioForeignPrivDirID;
    long ioForeignPrivInfo1;
    long ioForeignPrivInfo2;
    long ioForeignPrivInfo3;
    long ioForeignPrivInfo4;
};
typedef struct ForeignPrivParam ForeignPrivParam;
typedef ForeignPrivParam * ForeignPrivParamPtr;
```

Fields
`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`ioFiller21`
Reserved.

`ioFiller22`
Reserved.

`ioForeignPrivBuffer`
A pointer to a buffer containing access-control information about the foreign file system.

`ioForeignPrivActCount`
The size of the buffer pointed to by the `ioForeignPrivBuffer` field.

`ioForeignPrivReqCount`
The amount of the buffer pointed to by the `ioForeignPrivBuffer` field that was actually used to hold data.

`ioFiller23`
Reserved.

`ioForeignPrivDirID`
The parent directory ID of the foreign file or directory.

`ioForeignPrivInfo1`
A long word that may contain privileges data.

`ioForeignPrivInfo2`
A long word that may contain privileges data.

`ioForeignPrivInfo3`
A long word that may contain privileges data.

`ioForeignPrivInfo4`
A long word that may contain privileges data.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

FSCatalogBulkParam

Defines a parameter block used to retrieve catalog information in bulk on HFS Plus volumes.

```
struct FSCatalogBulkParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    Boolean containerChanged;
    UInt8 reserved;
    FSIteratorFlags iteratorFlags;
    FSIterator iterator;
    const FSRef * container;
    ItemCount maximumItems;
    ItemCount actualItems;
    FSCatalogInfoBitmap whichInfo;
    FSCatalogInfo * catalogInfo;
    FSRef * refs;
    FSSpec * specs;
    HFSUniStr255 * names;
    const FSSearchParams * searchParams;
};
typedef struct FSCatalogBulkParam FSCatalogBulkParam;
typedef FSCatalogBulkParam * FSCatalogBulkParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`containerChanged`
: A Boolean value indicating whether or not the container has changed since the last call to `PBGetCatalogInfoBulkSync` or `PBGetCatalogInfoBulkAsync`.

`reserved`
: Reserved.

`iteratorFlags`
: A set of flags which specifies how the iterator should iterate over the container. See "Iterator Flags" (page 924) for the meaning of the constants used here.

`iterator`
: A catalog iterator.

`container`
: An `FSRef` for the directory or volume to iterate over.

`maximumItems`
: The maximum number of items to return information about.

`actualItems`
: The actual number of items returned.

`whichInfo`
: A bitmap indicating which fields of the catalog information structure to return. See "Catalog Information Bitmap Constants" (page 891) for the bits defined for this field.

`catalogInfo`
: A pointer to an array of catalog information structures. On input, you should pass a pointer to an array of `maximumItemsFSCatalogInfo` (page 826) structures. On return, `actualItems` structures will be filled out with the information requested in the `whichInfo` field. If you do not wish any catalog information to be returned, pass a `NULL` pointer in this field and pass the constant `kFSCatInfoNone` in the `whichInfo` field.

`refs`
: A pointer to an array of `FSRef` structures. On input, you should pass a pointer to `maximumItemsFSRef` structures. On return, `actualItems` structures will be filled out. If you do not wish any `FSRef` structures to be returned, pass a `NULL` pointer in this field.

`specs`
: A pointer to an array of `FSSpec` structures. On input, you should pass a pointer to `maximumItems` file system specifications. On return, `actualItemsFSSpec` structures will be filled in. If you do not wish any `FSSpec` information to be returned, pass a `NULL` pointer in this field.

`names`
: A pointer to an array of Unicode names. On input, you should pass a pointer to an array of `maximumItemsHFSUniStr255` structures. On return, `actualItems` structures will contain Unicode names. If you do not wish any file or directory names to be returned, pass a `NULL` pointer in this field.

`searchParams`
: A pointer to an `FSSearchParams` (page 839) structure, specifying the values to match against.

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

Declared In
`Files.h`

FSCatalogInfo

Holds basic information about a file or directory.

```
struct FSCatalogInfo {
    UInt16 nodeFlags;
    FSVolumeRefNum volume;
    UInt32 parentDirID;
    UInt32 nodeID;
    UInt8 sharingFlags;
    UInt8 userPrivileges;
    UInt8 reserved1;
    UInt8 reserved2;
    UTCDateTime createDate;
    UTCDateTime contentModDate;
    UTCDateTime attributeModDate;
    UTCDateTime accessDate;
    UTCDateTime backupDate;
    UInt32 permissions[4];
    UInt8 finderInfo[16];
    UInt8 extFinderInfo[16];
    UInt64 dataLogicalSize;
    UInt64 dataPhysicalSize;
    UInt64 rsrcLogicalSize;
    UInt64 rsrcPhysicalSize;
    UInt32 valence;
    TextEncoding textEncodingHint;
};
typedef struct FSCatalogInfo FSCatalogInfo;
typedef FSCatalogInfo * FSCatalogInfoPtr;
```

Fields

`nodeFlags`
Node flags. This field has two defined bits that indicate whether an object is a file or folder, and whether a file is locked (constants `kFSNodeIsDirectoryMask` and `kFSNodeLockedMask`). See "Catalog Information Node Flags" (page 894) for the values you can use here.

`volume`
The object's volume reference.

`parentDirID`
The ID of the directory that contains the given object. The root directory of a volume always has ID `fsRtDirID` (2); the parent of the root directory is ID `fsRtParID` (1). Note that there is no object with ID `fsRtParID`; this is merely used when the File Manager is asked for the parent of the root directory.

`nodeID`
The file or directory ID.

`sharingFlags`
The object's sharing flags. See "Catalog Information Sharing Flags " (page 896) for the meaning of the bits defined for this field.

`userPrivileges`
The user's effective AFP privileges (same as `ioACUser` in the old `HFileInfo` and `DirInfo` structures). See "User Privileges Constants" (page 930).

`reserved1`
Reserved.

`reserved2`
Reserved.

`createDate`
The date and time of the creation of the object. Note that file systems other than AFP, HFS and HFS Plus do not generally support creation dates. For file systems which do not support creation dates, `FSGetCatalogInfo`, `PBGetCatalogInfoSync`, and `PBGetCatalogInfoAsync` return 0 in this field.

`contentModDate`
The date and time that the data or resource fork was last modified.

`attributeModDate`
The date and time that an attribute of the object (such as a fork other than the data or resource fork) was last modified.

`accessDate`
The date and time that the object was last accessed. The Mac OS 9 File Manager does not automatically update the `accessDate` field; it exists primarily for use by other operating systems (notably Mac OS X).

`backupDate`
The date and time of the object's last backup. This field is not updated by the File Manager a backup utility may use this field if it wishes. Note that file systems other than AFP, HFS and HFS Plus do not generally support backup dates. For file systems which do not support backup dates, `FSGetCatalogInfo`, `PBGetCatalogInfoSync`, and `PBGetCatalogInfoAsync` return 0 in this field.

`permissions`
User and group permission information. The Mac OS 8 and 9 File Manager does not use or enforce this permission information. It could be used by a file server program or other operating system (primarily Mac OS X). In Mac OS X, this array contains the file system permissions of the returned item. To use this information, coerce the parameter to a `FSPermissionInfo` (page 836) structure.

`finderInfo`
Basic Finder information for the object. This information is available in the catalog information, instead of in a named fork, for historical reasons. The File Manager does not interpret the contents of these fields. To use this information, coerce the parameter to a `FileInfo` (page 2258) or `FolderInfo` (page 2255) structure.

`extFinderInfo`
Extended Finder information for the object. This information is available in the catalog information, instead of in a named fork, for historical reasons. The File Manager does not interpret the contents of these fields. To use this information, coerce the parameter to an `ExtendedFileInfo` (page 2257) or `ExtendedFolderInfo` (page 2254) structure.

`dataLogicalSize`
The size of the data fork in bytes (the fork's logical size). The information in this field is only valid for files do not rely upon the value returned in this field for folders.

`dataPhysicalSize`
The amount of disk space, in bytes, occupied by the data fork (the fork's physical size). The information in this field is only valid for files do not rely upon the value returned in this field for folders.

`rsrcLogicalSize`
The size of the resource fork (the fork's logical size). The information in this field is only valid for files do not rely upon the value returned in this field for folders.

`rsrcPhysicalSize`

The amount of disk space occupied by the resource fork (the fork's physical size). The information in this field is only valid for files do not rely upon the value returned in this field for folders.

`valence`

For folders only, the number of items (files plus directories) contained within the directory. For files, it is set to zero. Many volume formats do not store a field containing a directory's valence. For those volume formats, this field is very expensive to compute. Think carefully before you ask the File Manager to return this field.

`textEncodingHint`

The `textEncodingHint` field is used in conjunction with the Unicode filename of the object. It is an optional hint that can be used by the volume format when converting the Unicode to some other encoding. For example, HFS Plus stores this value and uses it when converting the name to a Mac OS encoding, such as when the name is returned by `PBGetCatInfoSync` or `PBGetCatInfoAsync`. As another example, HFS volumes use this value to convert the Unicode name to a Mac OS encoded name stored on disk. If the entire Unicode name can be converted to a single Mac OS encoding, then that encoding should be used as the `textEncodingHint`; otherwise, a text encoding corresponding to the first characters of the name will probably provide the best user experience.

If a `textEncodingHint` is not supplied when a file or directory is created or renamed, the volume format will use a default value. This default value may not be the best possible choice for the given filename. Whenever possible, a client should supply a `textEncodingHint`.

Discussion
The `FSCatalogInfoBitmap` type is used to indicate which fields of the `FSCatalogInfo` should be set or retrieved. If the bit corresponding to a particular field is not set, then that field is not changed if the `FSCatalogInfo` is an output parameter, and that field is ignored if the `FSCatalogInfo` is an input parameter.

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

Declared In
`Files.h`

FSCatalogInfoBitmap

Describes which fields of the `FSCatalogInfo` structure you wish to retrieve or set.

```
typedef UInt32 FSCatalogInfoBitmap;
```

Discussion
If the bit corresponding to a particular field is not set in the bitmap, then that field is not changed in the `FSCatalogInfo` structure if it is an output parameter, and that field is ignored if the `FSCatalogInfo` structure is an input parameter. See "Catalog Information Bitmap Constants" (page 891) for a description of the constants you should use with this data type.

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

Declared In
`Files.h`

FSEjectStatus

```
typedef UInt32 FSEjectStatus;
```

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

Declared In
`Files.h`

FSFileOperationClientContext

Specifies user-defined data and callbacks associated with an asynchronous file operation.

```
struct FSFileOperationClientContext {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
};
typedef struct FSFileOperationClientContext FSFileOperationClientContext;
```

Fields

`version`
: The version number of the structure; this field should always contain 0.

`info`
: A generic pointer to your user-defined data. This pointer is passed back to your application when you check the status of the file operation. There are two ways you can ask the File Manager for status information about a file operation: by supplying a status callback function when you start the operation, or by calling a file operation status function directly.

`retain`
: An optional callback function that the File Manager can use to retain the user-defined data specified in the `info` parameter. If your data is a Core Foundation object, you can simply specify the function `CFRetain`. If no callback is needed, set this field to `NULL`.

`release`
: An optional callback function that the File Manager can use to release the user-defined data specified in the `info` parameter. If your data is a Core Foundation object, you can simply specify the function `CFRelease`. If no callback is needed, set this field to `NULL`.

`copyDescription`
: An optional callback function that the File Manager can use to create a descriptive string representation of your user-defined data for debugging purposes. If no callback is needed, set this field to `NULL`.

Discussion
You supply a client context when calling functions such as `FSCopyObjectAsync` (page 477) or `FSMoveObjectAsync` (page 511) that start an asynchronous copy or move operation.

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

Declared In
`Files.h`

FSFileOperationRef

Defines an opaque type that represents an asynchronous file operation.

```
typedef struct __FSFileOperation * FSFileOperationRef;
```

Discussion
You supply a file operation object when calling functions such as `FSCopyObjectAsync` (page 477) or `FSMoveObjectAsync` (page 511) to start an asynchronous copy or move operation. You can also use a file operation object to check the status of a file operation or to cancel the operation.

To perform an asynchronous file operation:

1. Create a file operation object using the function `FSFileOperationCreate` (page 488).
2. Pass the object to the function `FSFileOperationScheduleWithRunLoop` (page 489) to schedule the operation.
3. Pass the object to one of the asynchronous file operation functions to start the operation.

The `FSFileOperationRef` opaque type is a standard Core Foundation data type. It 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
`Files.h`

FSForkCBInfoParam

Defines a parameter block used by low-level HFS Plus fork control block functions.

```
struct FSForkCBInfoParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    SInt16 desiredRefNum;
    SInt16 volumeRefNum;
    SInt16 iterator;
    SInt16 actualRefNum;
    FSRef * ref;
    FSForkInfo * forkInfo;
    HFSUniStr255 * forkName;
};
typedef struct FSForkCBInfoParam FSForkCBInfoParam;
typedef FSForkCBInfoParam * FSForkCBInfoParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`desiredRefNum`
A fork reference number.

`volumeRefNum`
The volume reference number of the volume to match, or zero to match all volumes.

`iterator`
An iterator. Set to zero to start iteration.

`actualRefNum`
On return, the actual fork reference number found.

`ref`
A pointer to an `FSRef` for the specified fork.

`forkInfo`
A pointer to a fork information structure, `FSForkInfo` (page 832).

`forkName`
A pointer to the fork's Unicode name.

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

Declared In
`Files.h`

FSForkInfo

Contains information about an open fork.

```
struct FSForkInfo {
    SInt8 flags;
    SInt8 permissions;
    FSVolumeRefNum volume;
    UInt32 reserved2;
    UInt32 nodeID;
    UInt32 forkID;
    UInt64 currentPosition;
    UInt64 logicalEOF;
    UInt64 physicalEOF;
    UInt64 process;
};
typedef struct FSForkInfo FSForkInfo;
typedef FSForkInfo * FSForkInfoPtr;
```

Fields

`flags`
Flags describing the status of the fork. See "FCB Flags" (page 906) for a description of the bits in this field.

`permissions`
User and group permission information.

`volume`
A volume specification. This can be a volume reference number, drive number, or 0 for the default volume.

`reserved2`
Reserved.

`nodeID`
The file or directory ID of the file or directory with which the fork is associated.

`forkID`
The fork ID.

`currentPosition`
The current position within the fork.

`logicalEOF`
The logical size of the fork.

`physicalEOF`
The physical size of the fork.

`process`
The process which opened the fork.

Discussion
This data type is used in the `forkInfo` parameter of the `FSGetForkCBInfo` function, and in the `forkInfo` field of the `FSForkCBInfoParam` parameter block passed to the `PBGetForkCBInfoSync` and `PBGetForkCBInfoAsync` functions. When these functions return, the `FSForkInfo` structure contains information about the specified open fork.

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

Declared In
`Files.h`

FSForkIOParam

Defines a parameter block used by low-level HFS Plus fork I/O functions.

```
struct FSForkIOParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    void * reserved1;
    SInt16 reserved2;
    SInt16 forkRefNum;
    UInt8 reserved3;
    SInt8 permissions;
    const FSRef * ref;
    Ptr buffer;
    UInt32 requestCount;
    UInt32 actualCount;
    UInt16 positionMode;
    SInt64 positionOffset;
    FSAllocationFlags allocationFlags;
    UInt64 allocationAmount;
    UniCharCount forkNameLength;
    const UniChar * forkName;
    CatPositionRec forkIterator;
    HFSUniStr255 * outForkName;
};
typedef struct FSForkIOParam FSForkIOParam;
typedef FSForkIOParam * FSForkIOParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
: A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`reserved1`
: Reserved.

`reserved2`
: Reserved.

`forkRefNum`
: A reference number for a fork.

`reserved3`
: Reserved.

`permissions`
: The desired type of access to the specified fork. See "File Access Permission Constants" (page 908) for a description of the types of access that you can request.

`ref`
: An `FSRef` for the file or directory to open.

`buffer`
: A pointer to a data buffer.

`requestCount`
: The number of bytes requested for the given operation.

`actualCount`
: The actual number of bytes completed by the call.

`positionMode`
: A constant indicating the base location within the file for the start of the operation. See "Position Mode Constants" (page 928) for the meaning of the constants you can use in this field.

`positionOffset`
: The offset from the base location specified in the `positionMode` offset for the start of the operation.

`allocationFlags`
: A set of bit flags used by the `FSAllocateFork` (page 470) function to control how space is allocated. See "Allocation Flags" (page 887) for a description of the defined flags.

`allocationAmount`
: For the `FSAllocateFork` (page 470) function, the amount of space, in bytes, to allocate.

`forkNameLength`
: The length of the file or directory name passed in the `forkName` field, in Unicode characters.

`forkName`
: A pointer to the file or directory's Unicode name. This field is an input parameter functions which return the file or directory name in the parameter block use the `outForkName` field.

`forkIterator`
: A fork iterator.

`outForkName`

A pointer to the file or directory's Unicode name this is an output parameter. For functions which require the file or directory name as an input argument, you should pass a pointer to that name in the `forkName` field and pass the length of the name in the `forkNameLength` field.

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

Declared In
`Files.h`

FSIterator

Refers to a position within the catalog, used when iterating over files and folders in a directory.

```
typedef struct OpaqueFSIterator * FSIterator;
```

Discussion
This data type is like a file reference number because it maintains state internally to the File Manager and must be explicitly opened and closed.

An `FSIterator` is returned by `FSOpenIterator` and is passed as input to `FSGetCatalogInfoBulk`, `FSCatalogSearch` and `FSCloseIterator`.

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

Declared In
`Files.h`

FSMountStatus

```
typedef UInt32 FSMountStatus;
```

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

Declared In
`Files.h`

FSPermissionInfo

```
struct FSPermissionInfo {
    UInt32 userID;
    UInt32 groupID;
    UInt8 reserved1;
    UInt8 userAccess;
    UInt16 mode;
    UInt32 reserved2;
};
typedef struct FSPermissionInfo FSPermissionInfo;
```

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

Declared In
`Files.h`

FSRangeLockParam

Defines a parameter block for use with 64-bit range locking functions.

```
struct FSRangeLockParam {
    QElemPtr qLink;
    SInt16 qType;
    SInt16 ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    FSIORefNum forkRefNum;
    UInt64 requestCount;
    UInt16 positionMode;
    SInt64 positionOffset;
    UInt64 rangeStart;
};
typedef struct FSRangeLockParam FSRangeLockParam;
```

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

Declared In
`Files.h`

FSRangeLockParamPtr

Defines a pointer to a range lock parameter block.

```
typedef FSRangeLockParam *FSRangeLockParamPtr;
```

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

Declared In
`Files.h`

FSRef

Identifies a directory or file, including a volume's root directory.

```
struct FSRef {
    UInt8 hidden[80];
};
typedef struct FSRef FSRef;
typedef FSRef * FSRefPtr;
```

Discussion
This data type's purpose is similar to an `FSSpec` except that an `FSRef` is completely opaque. An `FSRef` contains whatever information is needed to find the given object; the internal structure of an `FSRef` is likely to vary based on the volume format, and may vary based on the particular object being identified.

The client of the File Manager cannot examine the contents of an `FSRef` to extract information about the parent directory or the object's name. Similarly, an `FSRef` cannot be constructed directly by the client; the `FSRef` must be constructed and returned via the File Manager. There is no need to call the File Manager to dispose an `FSRef`.

To determine the volume, parent directory and name associated with an `FSRef`, or to get an equivalent `FSSpec`, use the `FSGetCatalogInfo` call.

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

Declared In
`Files.h`

FSRefParam

Defines a parameter block used by low-level HFS Plus functions.

```
struct FSRefParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    ConstStringPtr ioNamePtr;
    short ioVRefNum;
    SInt16 reserved1;
    UInt8 reserved2;
    UInt8 reserved3;
    const FSRef * ref;
    FSCatalogInfoBitmap whichInfo;
    FSCatalogInfo * catInfo;
    UniCharCount nameLength;
    const UniChar * name;
    long ioDirID;
    FSSpec * spec;
    FSRef * parentRef;
    FSRef * newRef;
    TextEncoding textEncodingHint;
    HFSUniStr255 * outName;
};
typedef struct FSRefParam FSRefParam;
typedef FSRefParam * FSRefParamPtr;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`

The queue type. This field is used internally by the File Manager.

`ioTrap`

The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`

The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`

A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`

The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`

A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—you should set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
A volume reference number, or 0 for the default volume.

`reserved1`
Reserved.

`reserved2`
Reserved.

`reserved3`
Reserved.

`ref`
The `FSRef` describing the file or directory which is the target of the call.

`whichInfo`
An `FSCatalogInfoBitmap` which describes the fields of the catalog information structure passed in the `catInfo` field which are to be retrieved or set.

`catInfo`
A catalog information structure containing information about the specified file or directory.

`nameLength`
The length of the file or directory's name, for the `PBCreateSync`, `PBCreateAsync`, `PBRenameSync`, and `PBRenameAsync` functions.

`name`
A pointer to the file or directory's Unicode name, for the `PBCreateSync`, `PBCreateAsync`, `PBRenameSync`, and `PBRenameAsync` functions.

`ioDirID`
The directory ID of the specified file or directory's parent directory.

`spec`
The target or source `FSRef`.

`parentRef`
The secondary or destination `FSRef`. (Or the ref of the directory to move another file or directory to).

`newRef`
The output `FSRef` (ie, a new `FSRef`).

`textEncodingHint`
A text encoding hint for the file or directory's Unicode name, used by the `PBMakeFSRefSync`, `PBMakeFSRefAsync`, `PBRenameSync`, and `PBRenameAsync` functions.

`outName`
On output, a pointer to the Unicode name of the file or directory, used by the `PBGetCatalogInfoSync` and `PBGetCatalogInfoAsync` functions.

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

Declared In
`Files.h`

FSSearchParams

Describes the search criteria for a catalog information search.

```
struct FSSearchParams {
    Duration searchTime;
    OptionBits searchBits;
    UniCharCount searchNameLength;
    const UniChar * searchName;
    FSCatalogInfo * searchInfo1;
    FSCatalogInfo * searchInfo2;
};
typedef struct FSSearchParams FSSearchParams;
typedef FSSearchParams * FSSearchParamsPtr;
```

Fields

`searchTime`

A Time Manager duration for the duration of the search. If you specify a non-zero value in this field, the search may terminate after the specified time, even if the maximum number of requested objects has not been returned and the entire catalog has not been scanned.

If this value is negative, the time is interpreted in microseconds; if positive, it is interpreted as milliseconds. If `searchTime` is zero, there is no time limit on the search.

`searchBits`

A set of bits specifying which catalog information fields to search on. See "Catalog Search Constants" (page 899) for the constants which you can use here.

`searchNameLength`

The length of the Unicode name to search by.

`searchName`

A pointer to the Unicode name to search by.

`searchInfo1`

An `FSCatalogInfo` (page 826) structure which specifies the values and lower bounds of a search.

`searchInfo2`

A `FSCatalogInfo` (page 826) structure which specifies the masks and upper bounds of a search.

Discussion

Used by `FSCatalogSearch`, `PBCatalogSearchSync`, and `PBCatalogSearchAsync` to specify the criteria for a catalog search.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

FSSpec

Specifies the name and location of a file or directory.

```
struct FSSpec {
    short vRefNum;
    long parID;
    StrFileName name;
};
typedef struct FSSpec FSSpec;
typedef FSSpec * FSSpecPtr;
```

Fields

`vRefNum`

The volume reference number of the volume containing the specified file or directory.

`parID`

The parent directory ID of the specified file or directory (the directory ID of the directory containing the given file or directory).

`name`

The name of the specified file or directory. In Carbon, this name must be a leaf name; the name cannot contain a semicolon.

Discussion

The `FSSpec` structure can describe only a file or a directory, not a volume. A volume can be identified by its root directory, although the system software never uses an `FSSpec` structure to describe a volume. The directory ID of the root's parent directory is `fsRtParID`. The name of the root directory is the same as the name of the volume.

If you need to convert a file specification into an `FSSpec` structure, call the function `FSMakeFSSpec` (page 505) . Do not fill in the fields of an `FSSpec` structure yourself.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

FSSpecArrayPtr

Defines a pointer to an array of `FSSpec` structures.

```
typedef FSSpecPtr FSSpecArrayPtr;
```

Discussion

See `FSSpec` (page 840).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

FSUnmountStatus

```
typedef UInt32 FSUnmountStatus;
```

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

Declared In
`Files.h`

FSVolumeEjectUPP

```
typedef FSVolumeEjectProcPtr FSVolumeEjectUPP;
```

Discussion
For more information, see the description of the `FSVolumeEjectProcPtr` (page 792) callback function.

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

Declared In
`Files.h`

FSVolumeInfo

Used when getting or setting information about a volume.

```
struct FSVolumeInfo {
    UTCDateTime createDate;
    UTCDateTime modifyDate;
    UTCDateTime backupDate;
    UTCDateTime checkedDate;
    UInt32 fileCount;
    UInt32 folderCount;
    UInt64 totalBytes;
    UInt64 freeBytes;
    UInt32 blockSize;
    UInt32 totalBlocks;
    UInt32 freeBlocks;
    UInt32 nextAllocation;
    UInt32 rsrcClumpSize;
    UInt32 dataClumpSize;
    UInt32 nextCatalogID;
    UInt8 finderInfo[32];
    UInt16 flags;
    UInt16 filesystemID;
    UInt16 signature;
    UInt16 driveNumber;
    short driverRefNum;
};
typedef struct FSVolumeInfo FSVolumeInfo;
typedef FSVolumeInfo * FSVolumeInfoPtr;
```

Fields

`createDate`
: The date and time the volume was created. A value of 0 means that the volume creation date is unknown.

`modifyDate`
: The last time when the volume was modified in any way. A value of 0 means “never” or “unknown.

`backupDate`
: Indicates when the volume was last backed up. This field is for use by backup utilities. A value of 0 means “never” or “unknown.

`checkedDate`
: The last date and time that the volume was checked for consistency. A value of 0 means “never” or “unknown.

`fileCount`
: The total number of files on the volume, or 0 if unknown.

`folderCount`
: The total number of folders on the volume, or 0 if unknown. Note that no root directory counts.

`totalBytes`
: The size of the volume in bytes.

`freeBytes`
: The number of bytes of free space on the volume.

`blockSize`
: The size of an allocation block, in bytes. This field is only appropriate for volume formats (such as HFS and HFS Plus) that allocate space in fixed-size pieces; other volume formats may not have a similar concept, and may set this field to zero.

`totalBlocks`
: The total number of allocation blocks on the volume. This field is only appropriate for volume formats (such as HFS and HFS Plus) that allocate space in fixed-size pieces; other volume formats may not have a similar concept, and may set this field to zero.

`freeBlocks`
: The number of unused allocation blocks on the volume. This field is only appropriate for volume formats (such as HFS and HFS Plus) that allocate space in fixed-size pieces; other volume formats may not have a similar concept, and may set this field to zero.

`nextAllocation`
: A hint for where to start searching for free space during an allocation. This field is only appropriate for volume formats (such as HFS and HFS Plus) that allocate space in fixed-size pieces; other volume formats may not have a similar concept, and may set this field to zero.

`rsrcClumpSize`
: Default resource fork clump size. When a fork is automatically grown as it is written, the File Manager attempts to allocate space that is a multiple of the clump size. This field is zero for volume formats that don't support the notion of a clump size.

`dataClumpSize`
: Default data fork clump size. When a fork is automatically grown as it is written, the File Manager attempts to allocate space that is a multiple of the clump size. This field is zero for volume formats that don't support the notion of a clump size.

`nextCatalogID`
: The next unused catalog node ID. Some volume formats (such as HFS and HFS Plus) use a monotonically increasing number for the catalog node ID (i.e. File ID or Directory ID) of newly created files and directories. For those volume formats, the `nextCatalogID` is the next file/directory ID that will be assigned. For other volume formats, this field will be zero.

`finderInfo`
: Information used by Finder, such as the Directory ID of the System Folder. Some volume formats do not support Finder information for a volume and will set this field to all zeroes.

`flags`
: This field contains bit flags holding information about the volume. See "Volume Information Flags" (page 940) for the attribute constants you can use here.

`filesystemID`
: Identifies the filesystem implementation that is handling the volume; this is zero for HFS and HFS Plus volumes.

`signature`
: This field is used to distinguish between volume formats supported by a single filesystem implementation.

`driveNumber`
: The drive number for the drive (drive queue element) associated with the volume. Mac OS X does not support drive numbers; in Mac OS X, the File Manager always returns a value of 1 in this field.

`driverRefNum`
: The driver reference number for the drive (drive queue element) associated with the volume.

Discussion
This structure contains information about a volume as a whole information about a volume's root directory would use the `FSCatalogInfo` (page 826) structure.

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

Declared In
`Files.h`

FSVolumeInfoBitmap

Describes which fields of the `FSVolumeInfo` structure you wish to retrieve or set.

```
typedef UInt32 FSVolumeInfoBitmap;
```

Discussion
If the bit corresponding to a particular field is not set in the bitmap, then that field is not changed in the `FSVolumeInfo` structure if it is an output parameter, and that field is ignored if the `FSVolumeInfo` structure is an input parameter. See "Volume Information Bitmap Constants" (page 938) for a description of the constants you should use with this data type.

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

Declared In
`Files.h`

FSVolumeInfoParam

Defines a parameter block used by low-level HFS Plus volume manipulation functions.

```
struct FSVolumeInfoParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    FSVolumeRefNum ioVRefNum;
    UInt32 volumeIndex;
    FSVolumeInfoBitmap whichInfo;
    FSVolumeInfo * volumeInfo;
    HFSUniStr255 * volumeName;
    FSRef * ref;
};
typedef struct FSVolumeInfoParam FSVolumeInfoParam;
typedef FSVolumeInfoParam * FSVolumeInfoParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
: A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a volume name. This field is unused.

`ioVRefNum`
: The volume reference number.

`volumeIndex`
: The volume index. If this field is 0, the value in the `ioVRefNum` field only is used to identify the target volume.

`whichInfo`
: A bitmap indicating which volume information fields to retrieve or set in the `FSVolumeInfo` (page 842) structure passed in the `volumeInfo` field. See "Volume Information Bitmap Constants" (page 938) for the meaning of the bits in this field.

`volumeInfo`
: A pointer to a volume information structure containing the requested volume information on return, or the new values of the volume information to set on input. See `FSVolumeInfo` (page 842).

`volumeName`
: On output, a pointer to the volume's name.

`ref`
: A pointer to an `FSRef` for the specified volume's root directory.

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

Declared In
`Files.h`

FSVolumeMountUPP

```
typedef FSVolumeMountProcPtr FSVolumeMountUPP;
```

Discussion
For more information, see the description of the `FSVolumeMountProcPtr` (page 792) callback function.

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

Declared In
`Files.h`

FSVolumeOperation

```
typedef struct OpaqueFSVolumeOperation * FSVolumeOperation;
```

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

Declared In
`Files.h`

FSVolumeRefNum

Identifies a particular mounted volume.

```
typedef SInt16 FSVolumeRefNum;
```

Discussion
This data type is the same as the 16-bit volume refnum previously passed in the `ioVRefNum` fields of a parameter block; this is simply a new type name for the old data type.

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

Declared In
`Files.h`

FSVolumeUnmountUPP

```
typedef FSVolumeUnmountProcPtr FSVolumeUnmountUPP;
```

Discussion
For more information, see the description of the `FSVolumeUnmountProcPtr` (page 793) callback function.

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

Declared In
`Files.h`

GetVolParmsInfoBuffer

Defines a volume attributes buffer, used by the `PBHGetVolParmsSync` and `PBHGetVolParmAsync` functions to return volume information.

```
struct GetVolParmsInfoBuffer {
    short vMVersion;
    long vMAttrib;
    Handle vMLocalHand;
    long vMServerAdr;
    long vMVolumeGrade;
    short vMForeignPrivID;
    long vMExtendedAttributes;
    void * vMDeviceID;
    UniCharCount vMMaxNameLength;
};
typedef struct GetVolParmsInfoBuffer GetVolParmsInfoBuffer;
```

Fields

`vMVersion`

The version number of the attributes buffer structure. Currently this field returns 1, 2, 3 or 4. Version 3 is introduced to support the HFS Plus APIs.

`vMAttrib`

A 32-bit quantity that encodes information about the volume attributes. See "Volume Attribute Constants" (page 931) for the meaning of the bits in this field.

`vMLocalHand`

A handle to private data for shared volumes. On creation of the VCB (right after mounting), this field is a handle to a 2-byte block of memory. The Finder uses this for its local window list storage, allocating and deallocating memory as needed. It is disposed of when the volume is unmounted. Your application should treat this field as reserved.

`vMServerAdr`

For AppleTalk server volumes, this field contains the internet address of an AppleTalk server volume. Your application can inspect this field to tell which volumes belong to which server; the value of this field is 0 if the volume does not have a server.

`vMVolumeGrade`

The relative speed rating of the volume. The scale used to determine these values is currently uncalibrated. In general, lower values indicate faster speeds. A value of 0 indicates that the volume's speed is unrated. The buffer version returned in the `vMVersion` field must be greater than 1 for this field to be meaningful.

`vMForeignPrivID`

An integer representing the privilege model supported by the volume. Currently two values are defined for this field: 0 represents a standard HFS or HFS Plus volume that might or might not support the AFP privilege model; `fsUnixPriv` represents a volume that supports the A/UX privilege model. The buffer version returned in the `vMVersion` field must be greater than 1 for this field to be meaningful.

`vMExtendedAttributes`

Contains bits that describe a volume's extended attributes. For this field to be meaningful, the `vMVersion` must be greater than 2. See "Extended Volume Attributes" (page 903) for the meaning of the bits in this field.

`vMDeviceID`

A device name identifying the device in /dev that corresponds to the volume. You can use this string to build a POSIX path to the device for use with IOKit APIs.

`vMMaxNameLength`

Discussion
Volumes that implement the HFS Plus APIs must use version 3 (or newer) of the `GetVolParmsInfoBuffer`. Volumes that don't implement the HFS Plus APIs may still implement version 3 of the `GetVolParmsInfoBuffer`. If the version of the `GetVolParmsInfoBuffer` is 2 or less, or the `bSupportsHFSPlusAPIs` bit is clear (zero), then the volume does not implement the HFS Plus APIs, and they are being emulated for that volume by the File Manager itself.

If a volume does not implement the HFS Plus APIs, and supports version 2 or earlier of the `GetVolParmsInfoBuffer`, it cannot itself describe whether it supports the `FSCatalogSearch` (page 472) or `FSExchangeObjects` calls. The compatibility layer will implement the `FSCatalogSearch` call if the volume supports the `PBCatSearch` call (i.e. the `bHasCatSearch` bit of `vMAttrib` is set). The compatibility layer will implement the `FSExchangeObjects` call if the volume supports `PBExchangeFiles` (i.e. the `bHasFileIDs` bit of `vMAttrib` is set).

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

Declared In
`Files.h`

HFileInfo

Defines a structure which holds catalog information about a file.

```
struct HFileInfo {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioFRefNum;
    SInt8 ioFVersNum;
    SInt8 filler1;
    short ioFDirIndex;
    SInt8 ioFlAttrib;
    SInt8 ioACUser;
    FInfo ioFlFndrInfo;
    long ioDirID;
    unsigned short ioFlStBlk;
    long ioFlLgLen;
    long ioFlPyLen;
    unsigned short ioFlRStBlk;
    long ioFlRLgLen;
    long ioFlRPyLen;
    unsigned long ioFlCrDat;
    unsigned long ioFlMdDat;
    unsigned long ioFlBkDat;
    FXInfo ioFlXFndrInfo;
    long ioFlParID;
    long ioFlClpSiz;
};
typedef struct HFileInfo HFileInfo;
```

Fields

`qLink`
: A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
: The queue type. This field is used internally by the File Manager.

`ioTrap`
: The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
: The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
: A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`ioFRefNum`
: The file reference number of an open file.

`ioFVersNum`
: A file version number. This field is no longer used. File version numbers are an artifact of the obsolete MFS, and are not supported on HFS volumes. You should always set this field to 0.

`filler1`
: Reserved.

`ioFDirIndex`
: A file and directory index. If this field contains a positive number, `PBGetCatInfoSync` and `PBGetCatInfoAsync` return information about the file or directory having that directory index in the directory specified by the `ioVRefNum` field. (If `ioVRefNum` contains a volume reference number, the specified directory is that volume's root directory.)

  If this field contains 0, `PBGetCatInfoSync` or `PBGetCatInfoAsync` returns information about the file or directory whose name is specified in the `ioNamePtr` field and that is located in the directory specified by the `ioVRefNum` field. (Once again, if `ioVRefNum` contains a volume reference number, the specified directory is that volume's root directory.)

  If this field contains a negative number, `PBGetCatInfoSync` or `PBGetCatInfoAsync` ignores the `ioNamePtr` field and returns information about the directory specified in the `ioDirID` field. If both `ioDirID` and `ioVRefNum` are set to 0, `PBGetCatInfoSync` or `PBGetCatInfoAsync` returns information about the current default directory.

`ioFlAttrib`
: File or directory attributes. See "File Attribute Constants" (page 914) for the meaning of the bits in this field.

`ioACUser`
: The user's access rights for the specified directory. See "User Privileges Constants" (page 930) for the meaning of the bits in this field.

`ioFlFndrInfo`
: Finder information.

`ioDirID`
: A directory ID or file ID. On input to `PBGetCatInfoSync` or `PBGetCatInfoAsync`, this field contains a directory ID (which is used only if the `ioFDirIndex` field is negative). On output, this field contains the file ID of the specified file.

`ioFlStBlk`
: The first allocation block of the data fork. This field contains 0 if the file's data fork is empty.

`ioFlLgLen`
: The logical length (logical end-of-file) of the data fork.

`ioFlPyLen`
: The physical length (physical end-of-file) of the data fork.

`ioFlRStBlk`
: The first allocation block of the resource fork.

`ioFlRLgLen`
: The logical length (logical end-of-file) of the resource fork.

`ioFlRPyLen`
: The physical length (physical end-of-file) of the resource fork.

`ioFlCrDat`
: The date and time of the file's creation, in seconds since midnight, January 1, 1904. However, on Mac OS X, if you set the creation date to a date between January 1, 1904 and January 1, 1970, it will be clipped to January 1, 1970, and that is the value which will be returned if you later try to retrieve the creation date.

  Note that file systems other than AFP, HFS and HFS Plus do not generally support creation dates.

`ioFlMdDat`
: The date and time of the last modification to the file, in seconds since midnight, January 1, 1904. However, on Mac OS X, if you set the modification date to a date between January 1, 1904 and January 1, 1970, it will be clipped to January 1, 1970.

`ioFlBkDat`
: The date and time that the file was last backed up, in seconds since midnight, January 1, 1904. However, on Mac OS X, if you set the backup date to a date between January 1, 1904 and January 1, 1970, it will be clipped to January 1, 1970.

  Note that file systems other than AFP, HFS and HFS Plus do not generally support backup dates.

`ioFlXFndrInfo`
: Additional Finder information.

`ioFlParID`
: The directory ID of the file's parent directory.

`ioFlClpSiz`
: The clump size to be used when writing the file if it's 0, the volume's clump size is used when the file is opened.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

HFileParam

Defines a parameter block used by low-level HFS functions for file creation, deletion, and information retrieval.

```
struct HFileParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioFRefNum;
    SInt8 ioFVersNum;
    SInt8 filler1;
    short ioFDirIndex;
    SInt8 ioFlAttrib;
    SInt8 ioFlVersNum;
    FInfo ioFlFndrInfo;
    long ioDirID;
    unsigned short ioFlStBlk;
    long ioFlLgLen;
    long ioFlPyLen;
    unsigned short ioFlRStBlk;
    long ioFlRLgLen;
    long ioFlRPyLen;
    unsigned long ioFlCrDat;
    unsigned long ioFlMdDat;
};
typedef struct HFileParam HFileParam;
typedef HFileParam * HFileParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`ioFRefNum`
The file reference number of an open file.

`ioFVersNum`
A file version number. This field is no longer used. File version numbers are an artifact of the obsolete MFS, and are not supported on HFS volumes. You should always set this field to 0.

`filler1`
Reserved.

`ioFDirIndex`
A directory index for use with the `PBHGetFInfoSync` (page 683) and `PBHGetFInfoAsync` (page 682) functions.

`ioFlAttrib`
File attributes. See "File Attribute Constants" (page 914) for the meaning of the bits in this field.

`ioFlVersNum`
A file version number. This feature is no longer supported, and you must always set this field to 0.

`ioFlFndrInfo`
Information used by the Finder.

`ioDirID`
A directory ID.

`ioFlStBlk`
The first allocation block of the data fork. This field contains 0 if the file's data fork is empty.

`ioFlLgLen`
The logical length (logical end-of-file) of the data fork.

`ioFlPyLen`
The physical length (physical end-of-file) of the data fork.

`ioFlRStBlk`
The first allocation block of the resource fork. This field contains 0 if the file's resource fork is empty.

`ioFlRLgLen`
The logical length (logical end-of-file) of the resource fork.

`ioFlRPyLen`
The physical length (physical end-of-file) of the resource fork.

`ioFlCrDat`
The date and time of the file's creation, specified in seconds since midnight, January 1, 1904.

`ioFlMdDat`
The date and time of the last modification to the file, specified in seconds since midnight, January 1, 1904.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

HFSUniStr255

Used by the File Manager to return Unicode strings.

```
struct HFSUniStr255 {
    UInt16 length;
    UniChar unicode[255];
};
typedef struct HFSUniStr255 HFSUniStr255;
```

Fields

`length`

The number of unicode characters in the string.

`unicode`

The string, in unicode characters.

Discussion

This data type is a string of up to 255 16-bit Unicode characters, with a preceding 16-bit length (number of characters). Note that only the first length characters have meaningful values; the remaining characters may be set to arbitrary values. A caller should always assume that the entire structure will be modified, even if the actual string length is less than 255.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

HIOParam

Defines a parameter block used by low-level HFS I/O functions.

```
struct HIOParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioRefNum;
    SInt8 ioVersNum;
    SInt8 ioPermssn;
    Ptr ioMisc;
    Ptr ioBuffer;
    long ioReqCount;
    long ioActCount;
    short ioPosMode;
    long ioPosOffset;
};
typedef struct HIOParam HIOParam;
typedef HIOParam * HIOParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`ioRefNum`
The file reference number of an open file.

`ioVersNum`
: A version number. This field is no longer used and you should always set it to 0.

`ioPermssn`
: The access mode. See "File Access Permission Constants" (page 908).

`ioMisc`
: Depending on the function called, this field contains either a logical end-of-file, a new version number, a pointer to an access path buffer, or a pointer to a new pathname. Because `ioMisc` is of type `Ptr`, you'll need to perform type coercion to interpret the value of `ioMisc` correctly when it contains an end-of-file (a `LongInt` value) or version number (a `SignedByte` value).

`ioBuffer`
: A pointer to a data buffer into which data is written by `PBReadSync` and `PBReadAsync` calls, and from which data is read by `PBWriteSync` and `PBWriteAsync` calls.

`ioReqCount`
: The requested number of bytes to be read, written, or allocated.

`ioActCount`
: The number of bytes actually read, written, or allocated.

`ioPosMode`
: The positioning mode (base location) for setting the mark. Bits 0 and 1 of this field indicate how to position the mark; you can use the constants described in "Position Mode Constants" (page 928) to set or test their value.

  You can also use the constants described in "Cache Constants" (page 889) to indicate whether or not to cache the data.

`ioPosOffset`
: The offset to be used in conjunction with the base location specified in the `ioPosMode` field.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

HParamBlockRec

Describes the HFS parameter block.

```
union HParamBlockRec {
    HIOParam ioParam;
    HFileParam fileParam;
    HVolumeParam volumeParam;
    AccessParam accessParam;
    ObjParam objParam;
    CopyParam copyParam;
    WDParam wdParam;
    FIDParam fidParam;
    CSParam csParam;
    ForeignPrivParam foreignPrivParam;
};
typedef union HParamBlockRec HParamBlockRec;
typedef HParamBlockRec * HParmBlkPtr;
```

Fields

`ioParam`
A parameter block used by low-level HFS I/O functions. See `HIOParam` (page 855).

`fileParam`
A parameter block used by low-level HFS functions for file creation, deletion, and information retrieval. See `HFileParam` (page 852).

`volumeParam`
A parameter block used by low-level HFS volume manipulation functions. See `HVolumeParam` (page 859).

`accessParam`
A parameter block used by low-level HFS file and directory access rights manipulation functions. See `AccessParam` (page 795).

`objParam`
A parameter block used by low-level HFS user and group information functions. See `ObjParam` (page 865).

`copyParam`
A parameter block used by low-level HFS file copying functions. See `CopyParam` (page 806).

`wdParam`
A parameter block used by low-level HFS working directory functions. See `WDParam` (page 876).

`fidParam`
A parameter block used by low-level HFS file ID functions. See `FIDParam` (page 818).

`csParam`
A parameter block used by low-level HFS catalog search functions. See `CSParam` (page 807).

`foreignPrivParam`
A parameter block used by low-level HFS foreign privileges functions. See `ForeignPrivParam` (page 822).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

HVolumeParam

Defines a parameter block used by low-level HFS volume manipulation functions.

```
struct HVolumeParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    long filler2;
    short ioVolIndex;
    unsigned long ioVCrDate;
    unsigned long ioVLsMod;
    short ioVAtrb;
    unsigned short ioVNmFls;
    unsigned short ioVBitMap;
    unsigned short ioAllocPtr;
    unsigned short ioVNmAlBlks;
    unsigned long ioVAlBlkSiz;
    unsigned long ioVClpSiz;
    unsigned short ioAlBlSt;
    unsigned long ioVNxtCNID;
    unsigned short ioVFrBlk;
    unsigned short ioVSigWord;
    short ioVDrvInfo;
    short ioVDRefNum;
    short ioVFSID;
    unsigned long ioVBkUp;
    short ioVSeqNum;
    unsigned long ioVWrCnt;
    unsigned long ioVFilCnt;
    unsigned long ioVDirCnt;
    long ioVFndrInfo[8];
};
typedef struct HVolumeParam HVolumeParam;
typedef HVolumeParam * HVolumeParamPtr;
```

Fields

`qLink`
: A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
: The queue type. This field is used internally by the File Manager.

`ioTrap`
: The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
: The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
: A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`filler2`
: Reserved.

`ioVolIndex`
: A volume index for use with the `PBHGetVInfoSync` (page 690) and `PBHGetVInfoAsync` (page 686) functions.

`ioVCrDate`
: The date and time of the volume's initialization.

`ioVLsMod`
: The date and time the volume information was last modified. (This field is not changed when information is written to a file and does not necessarily indicate when the volume was flushed.

`ioVAtrb`
: The volume attributes. See "Volume Information Attribute Constants" (page 937) for the meanings of the bits in this field.

`ioVNmFls`
: The number of files in the root directory of the volume. For performance reasons, the Carbon File Manager does not return the number of files in this field; instead, it sets `ioVNmFls` to 0.

  To determine the number of files in the root directory of a volume in Carbon, call `PBGetCatInfoAsync` (page 648) or `PBGetCatInfoSync` (page 651) for the root directory. The number of files in the root directory is returned in the `ioDrNmFls` field.

`ioVBitMap`
: The first block of the volume bitmap.

`ioAllocPtr`
: The block at which the next new file starts. Used internally.

`ioVNmAlBlks`
: The number of allocation blocks.

`ioVAlBlkSiz`
: The size of allocation blocks.

`ioVClpSiz`
: The clump size.

`ioAlBlSt`
The first block in the volume map.

`ioVNxtCNID`
The next unused catalog node ID.

`ioVFrBlk`
The number of unused allocation blocks.

`ioVSigWord`
A signature word identifying the type of volume it's $D2D7 for MFS volumes and $4244 for volumes that support HFS calls.

`ioVDrvInfo`
The drive number of the drive containing the volume.

`ioVDRefNum`
For online volumes, the reference number of the I/O driver for the drive identified by the `ioVDrvInfo` field.

`ioVFSID`
The file-system identifier. It indicates which file system is servicing the volume it's zero for File Manager volumes and nonzero for volumes handled by an external file system.

`ioVBkUp`
The date and time the volume was last backed up; this is 0 if the volume has never been backed up.

`ioVSeqNum`
Used internally.

`ioVWrCnt`
The volume write count.

`ioVFilCnt`
The total number of files on the volume.

`ioVDirCnt`
The total number of directories (not including the root directory) on the volume.

`ioVFndrInfo`
Information used by the Finder.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

IOCompletionUPP

A universal procedure pointer to an application-defined completion function.

```
typedef IOCompletionProcPtr IOCompletionUPP;
```

Discussion
See `IOCompletionProcPtr` (page 794).

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

Declared In
`Files.h`

IOParam

Defines a parameter block used by low-level I/O functions.

```
struct IOParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioRefNum;
    SInt8 ioVersNum;
    SInt8 ioPermssn;
    Ptr ioMisc;
    Ptr ioBuffer;
    long ioReqCount;
    long ioActCount;
    short ioPosMode;
    long ioPosOffset;
};
typedef struct IOParam IOParam;
typedef IOParam * IOParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`ioRefNum`
: The file reference number of an open file.

`ioVersNum`
: A version number. This field is no longer used and you should always set it to 0.

`ioPermssn`
: The access mode. See "File Access Permission Constants" (page 908).

`ioMisc`
: Depending on the function called, this field contains either a new logical end-of-file (for the `PBGetEOFSync/ PBGetEOFAsync` and `PBSetEOFSync/ PBSetEOFAsync` functions), a new version number, or a pointer to a new pathname (for the `PBHRenameSync/ PBHRenameAsync` functions). Because `ioMisc` is of type `Ptr`, you'll need to perform type coercion to interpret the value of `ioMisc` correctly when it contains an end-of-file (a `LongInt` value) or version number (a `SignedByte` value).

`ioBuffer`
: A pointer to a data buffer into which data is written by `PBReadSync` and `PBReadAsync` calls; and from which data is read by `PBWriteSync` and `PBWriteAsync` calls.

`ioReqCount`
: The requested number of bytes to be read, written, or allocated.

`ioActCount`
: The number of bytes actually read, written, or allocated.

`ioPosMode`
: The positioning mode (base location) for positioning the file mark. Bits 0 and 1 of this field indicate how to position the mark; you can use the constants described in "Position Mode Constants" (page 928) to set or test their value.

  You can also use the constants described in "Cache Constants" (page 889) to indicate whether the data should be cached.

`ioPosOffset`
: The offset to be used in conjunction with the base location specified in the `ioPosMode` field.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

MultiDevParam

Defines a parameter block used by low-level functions in the classic Device Manager to access multiple devices.

```
struct MultiDevParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioMRefNum;
    SInt8 ioMVersNum;
    SInt8 ioMPermssn;
    Ptr ioMMix;
    short ioMFlags;
    Ptr ioSEBlkPtr;
};
typedef struct MultiDevParam MultiDevParam;
typedef MultiDevParam * MultiDevParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`ioMRefNum`
The driver reference number.

`ioMVersNum`
The slot version number.

`ioMPermssn`
Permissions.

`ioMMix`
Reserved.

`ioMFlags`
Flags specifying the number of additional fields. You should set the `fMulti` bit (bit 0) of this field and clear all of the other bits.

`ioSEBlkPtr`
A pointer to an external parameter block that is customized for the devices installed in the slot.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

ObjParam

Defines a parameter block used by low-level HFS user and group information functions.

```
struct ObjParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short filler7;
    short ioObjType;
    StringPtr ioObjNamePtr;
    long ioObjID;
};
typedef struct ObjParam ObjParam;
typedef ObjParam * ObjParamPtr;
```

Fields
`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`filler7`
: Reserved.

`ioObjType`
: A function code. The values passed in this field are determined by the function to which you pass this parameter block.

`ioObjNamePtr`
: A pointer to the returned creator/group name.

`ioObjID`
: The creator/group ID.

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

Not available to 64-bit applications.

Declared In
`Files.h`

ParamBlockRec

Describes the basic File Manager parameter block.

```
union ParamBlockRec {
    IOParam ioParam;
    FileParam fileParam;
    VolumeParam volumeParam;
    CntrlParam cntrlParam;
    SlotDevParam slotDevParam;
    MultiDevParam multiDevParam;
};
typedef union ParamBlockRec ParamBlockRec;
typedef ParamBlockRec * ParmBlkPtr;
```

Fields

`ioParam`

`fileParam`

`volumeParam`

`cntrlParam`

`slotDevParam`

`multiDevParam`

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

Declared In
`Files.h`

SlotDevParam

Defines a parameter block used by low-level functions in the classic Device Manager to access a single slot device.

```
struct SlotDevParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioSRefNum;
    SInt8 ioSVersNum;
    SInt8 ioSPermssn;
    Ptr ioSMix;
    short ioSFlags;
    SInt8 ioSlot;
    SInt8 ioID;
};
typedef struct SlotDevParam SlotDevParam;
typedef SlotDevParam * SlotDevParamPtr;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`ioSRefNum`
The driver reference number.

`ioSVersNum`
The slot version number.

`ioSPermssn`
Permissions.

`ioSMix`
Reserved.

`ioSFlags`
Flags determining the number of additional fields. You should clear all of the bits in this field.

`ioSlot`
The slot number.

`ioID`
The slot resource ID.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

VCB

Defines a volume control block.

```
struct VCB {
    QElemPtr qLink;
    short qType;
    short vcbFlags;
    unsigned short vcbSigWord;
    unsigned long vcbCrDate;
    unsigned long vcbLsMod;
    short vcbAtrb;
    unsigned short vcbNmFls;
    short vcbVBMSt;
    short vcbAllocPtr;
    unsigned short vcbNmAlBlks;
    long vcbAlBlkSiz;
    long vcbClpSiz;
    short vcbAlBlSt;
    long vcbNxtCNID;
    unsigned short vcbFreeBks;
    Str27 vcbVN;
    short vcbDrvNum;
    short vcbDRefNum;
    short vcbFSID;
    short vcbVRefNum;
    Ptr vcbMAdr;
    Ptr vcbBufAdr;
    short vcbMLen;
    short vcbDirIndex;
    short vcbDirBlk;
    unsigned long vcbVolBkUp;
    unsigned short vcbVSeqNum;
    long vcbWrCnt;
    long vcbXTClpSiz;
    long vcbCTClpSiz;
    unsigned short vcbNmRtDirs;
    long vcbFilCnt;
    long vcbDirCnt;
    long vcbFndrInfo[8];
    unsigned short vcbVCSize;
    unsigned short vcbVBMCSiz;
    unsigned short vcbCtlCSiz;
    unsigned short vcbXTAlBlks;
    unsigned short vcbCTAlBlks;
    short vcbXTRef;
    short vcbCTRef;
    Ptr vcbCtlBuf;
    long vcbDirIDM;
    short vcbOffsM;
};
typedef struct VCB VCB;
typedef VCB * VCBPtr;
```

Fields

`qLink`

A pointer to the next entry in the VCB queue.

`qType`

The queue type. When the volume is mounted and the VCB is created, this field is cleared. Thereafter, bit 7 of this field is set whenever a file on that volume is opened.

`vcbFlags`
: Volume flags. Bit 15 is set if the volume information has been changed by a File Manager call since the volume was last flushed by a `FlushVol` (page 466) call. See "Volume Control Block Flags" (page 935).

`vcbSigWord`
: The volume signature.

`vcbCrDate`
: The date and time of the volume's creation (initialization).

`vcbLsMod`
: The date and time of the volume's last modification. This is not necessarily when the volume was last flushed.

`vcbAtrb`
: The volume attributes.

`vcbNmFls`
: The number of files in the root directory of the volume.

`vcbVBMSt`
: The first block of the volume bitmap.

`vcbAllocPtr`
: The start block of the next allocation search. This field is used internally.

`vcbNmAlBlks`
: The number of allocation blocks in the volume.

`vcbAlBlkSiz`
: The allocation block size, in bytes. This value must always be a multiple of 512 bytes.

`vcbClpSiz`
: The default clump size.

`vcbAlBlSt`
: The first allocation block in the volume.

`vcbNxtCNID`
: The next unused catalog node ID (directory or file ID).

`vcbFreeBks`
: The number of unused allocation blocks on the volume.

`vcbVN`
: The volume name. Note that a volume name can occupy at most 27 characters; this is an exception to the normal file and directory name limit of 31 characters.

`vcbDrvNum`
: The drive number of the drive on which the volume is located. When a mounted drive is placed offline or ejected, this field is set to 0.

`vcbDRefNum`
: The driver reference number of the driver used to access the volume When a volume is ejected, this field is set to the previous value of the `vcbDrvNum` field (and hence is a positive number). When a volume is placed offline, this field is set to the negative of the previous value of the `vcbDrvNum` field (and hence is a negative number).

`vcbFSID`
: An identifier for the file system handling the volume it's zero for volumes handled by the File Manager and nonzero for volumes handled by other file systems.

`vcbVRefNum`
The volume reference number of the volume.

`vcbMAdr`
Used internally.

`vcbBufAdr`
Used internally.

`vcbMLen`
Used internally.

`vcbDirIndex`
Used internally.

`vcbDirBlk`
Used internally.

`vcbVolBkUp`
The date and time that the volume was last backed up.

`vcbVSeqNum`
Used internally.

`vcbWrCnt`
The volume write count.

`vcbXTClpSiz`
The clump size of the extents overflow file.

`vcbCTClpSiz`
The clump size of the catalog file.

`vcbNmRtDirs`
The number of directories in the root directory.

`vcbFilCnt`
The total number of files on the volume.

`vcbDirCnt`
The total number of directories on the volume.

`vcbFndrInfo`
Finder information.

`vcbVCSize`
Used internally.

`vcbVBMCSiz`
Used internally.

`vcbCtlCSiz`
Used internally.

`vcbXTAlBlks`
The size, in allocation blocks, of the extents overflow file.

`vcbCTAlBlks`
The size, in allocation blocks, of the catalog file.

`vcbXTRef`
The path reference number for the extents overflow file.

`vcbCTRef`
The path reference number for the catalog file.

`vcbCtlBuf`
A pointer to the extents and catalog caches.

`vcbDirIDM`
The directory last searched.

`vcbOffsM`
The offspring index at the last search.

Discussion
The volume control block queue is a standard operating system queue that's maintained in the system heap. It contains a volume control block for each mounted volume. A volume control block is a nonrelocatable block that contains volume-specific information.

Each time a volume is mounted, the File Manager reads its volume information from the master directory block and uses the information to build a new volume control block (VCB) in the volume control block queue (unless an ejected or offline volume is being remounted). The File Manager also creates a volume buffer in the system heap. When a volume is placed offline, its buffer is released. When a volume is unmounted, its VCB is removed from the VCB queue as well.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

VolMountInfoHeader

Defines a volume mounting information header structure for remote volumes.

```
struct VolMountInfoHeader {
    short length;
    VolumeType media;
};
typedef struct VolMountInfoHeader VolMountInfoHeader;
typedef VolMountInfoHeader * VolMountInfoPtr;
```

Fields
`length`
The length of the `VolMountInfoHeader` structure, which is the total length of the structure header described here, plus the variable-length location data which follows the header.

`media`
The volume type of the remote volume. The `AppleShareMediaType` represents an AppleShare volume.

If you are adding support for the programmatic mounting functions to a non-Macintosh file system, you should register a four-character identifier for your volumes with DTS.

Discussion
To mount a remote server, fill out an `VolMountInfoHeader` structure using the `PBGetVolMountInfo` function and then pass this structure to the `PBVolumeMount` function to mount the volume.

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

Declared In
`Files.h`

VolumeMountInfoHeader

Defines an extended volume mounting information header structure for remote volumes.

```
struct VolumeMountInfoHeader {
    short length;
    VolumeType media;
    short flags;
};
typedef struct VolumeMountInfoHeader VolumeMountInfoHeader;
typedef VolumeMountInfoHeader * VolumeMountInfoHeaderPtr;
```

Fields

`length`

The length of the `VolumeMountInfoHeader` structure, which is the total length of the structure header described here, plus the variable-length location data which follows the header.

`media`

The volume type of the remote volume. The `AppleShareMediaType` represents an AppleShare volume.

If you are adding support for the programmatic mounting functions to a non-Macintosh file system, you should register a four-character identifier for your volumes with DTS.

`flags`

The volume mount flags. See "Volume Mount Flags" (page 942).

Discussion
This volume mount info record supersedes the `VolMountInfoHeader` (page 872) structure; `VolMountInfoHeader` is included for compatibility. The `VolumeMountInfoHeader` record allows access to the volume mount flags by foreign filesystem writers.

To mount a remote server, fill out an `VolumeMountInfoHeader` structure using the `PBGetVolMountInfo` function and then pass this structure to the `PBVolumeMount` function to mount the volume.

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

Declared In
`Files.h`

VolumeParam

Defines a parameter block used by low-level volume manipulation functions.

```
struct VolumeParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    long filler2;
    short ioVolIndex;
    unsigned long ioVCrDate;
    unsigned long ioVLsBkUp;
    unsigned short ioVAtrb;
    unsigned short ioVNmFls;
    unsigned short ioVDirSt;
    short ioVBlLn;
    unsigned short ioVNmAlBlks;
    unsigned long ioVAlBlkSiz;
    unsigned long ioVClpSiz;
    unsigned short ioAlBlSt;
    unsigned long ioVNxtFNum;
    unsigned short ioVFrBlk;
};
typedef struct VolumeParam VolumeParam;
typedef VolumeParam * VolumeParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`filler2`
Reserved.

`ioVolIndex`
The volume index.

`ioVCrDate`
The date and time of the volume's initialization.

`ioVLsBkUp`
The date and time the volume information was last modified. (This field is not changed when information is written to a file and does not necessarily indicate when the volume was flushed.

`ioVAtrb`
The volume attributes. See "Volume Information Attribute Constants" (page 937) for the meanings of the bits in this field.

`ioVNmFls`
The number of files in the root directory.

`ioVDirSt`
The first block of the volume directory.

`ioVBlLn`
Length of directory in blocks.

`ioVNmAlBlks`
The number of allocation blocks.

`ioVAlBlkSiz`
The size of allocation blocks.

`ioVClpSiz`
The volume clump size.

`ioAlBlSt`
The first block in the volume map.

`ioVNxtFNum`
The next unused file number.

`ioVFrBlk`
The number of unused allocation blocks.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

VolumeType

Defines the "signature" of the file system.

```
typedef OSType VolumeType;
```

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

Declared In
`Files.h`

WDParam

Defines a parameter block used by low-level HFS working directory functions.

```
struct WDParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioWDCreated;
    short ioWDIndex;
    long ioWDProcID;
    short ioWDVRefNum;
    short filler10;
    long filler11;
    long filler12;
    long filler13;
    long ioWDDirID;
};
typedef struct WDParam WDParam;
typedef WDParam * WDParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`ioWDCreated`

`ioWDIndex`
: An index to working directories.

`ioWDProcID`

`ioWDVRefNum`
: The volume reference number for the working directory.

`filler10`
: Reserved.

`filler11`
: Reserved.

`filler12`
: Reserved.

`filler13`
: The working directory's directory ID.

`ioWDDirID`
: The working directory's directory ID.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

WDPBRec

Defines a working directory parameter block.

```
struct WDPBRec {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short filler1;
    short ioWDIndex;
    long ioWDProcID;
    short ioWDVRefNum;
    short filler2[7];
    long ioWDDirID;
};
typedef struct WDPBRec WDPBRec;
typedef WDPBRec * WDPBPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`filler1`
Reserved.

`ioWDIndex`
An index.

`ioWDProcID`
: An identifier that's used to distinguish between working directories set up by different users you should set `ioWDProcID` to your application's signature.

`ioWDVRefNum`
: The working directory's volume reference number.

`filler2`
: Reserved.

`ioWDDirID`
: The working directory's directory ID.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

XCInfoPBRec

Defines an extended catalog information parameter block.

```
struct XCInfoPBRec {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    ProcPtr ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    long filler1;
    StringPtr ioShortNamePtr;
    short filler2;
    short ioPDType;
    long ioPDAuxType;
    long filler3[2];
    long ioDirID;
};
typedef struct XCInfoPBRec XCInfoPBRec;
typedef XCInfoPBRec * XCInfoPBPtr;
```

Fields
`qLink`
: A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
: The queue type. This field is used internally by the File Manager.

`ioTrap`
: The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
: The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
: A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`filler1`
: Reserved; set this field to 0.

`ioShortNamePtr`
: A pointer to a Pascal string buffer, of a minimum 13 bytes, which holds the file or directory's short name (MS-DOS format name). This field is required and cannot be `NULL`.

`filler2`
: Reserved; set this field to 0.

`ioPDType`
: The ProDOS file type of the file or directory.

`ioPDAuxType`
: The ProDOS auxiliary type of the file or directory.

`filler3`
: Reserved; set this field to 0.

`ioDirID`
: A directory ID.

Discussion
The `PBGetXCatInfoSync` and `PBGetXCatInfoAsync` functions use this parameter block to return the short name and ProDOS information for files and directories.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

XIOParam

Defines an extended I/O parameter block structure.

```
struct XIOParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioRefNum;
    SInt8 ioVersNum;
    SInt8 ioPermssn;
    Ptr ioMisc;
    Ptr ioBuffer;
    long ioReqCount;
    long ioActCount;
    short ioPosMode;
    wide ioWPosOffset;
};
typedef struct XIOParam XIOParam;
typedef XIOParam * XIOParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`ioRefNum`
The file reference number of an open file.

`ioVersNum`
: A version number. This field is no longer used and you should always set it to 0.

`ioPermssn`
: The access mode. See "File Access Permission Constants" (page 908).

`ioMisc`
: Depending on the function called, this field contains either a logical end-of-file, a new version number, a pointer to an access path buffer, or a pointer to a new pathname. Because `ioMisc` is of type `Ptr`, you'll need to perform type coercion to interpret the value of `ioMisc` correctly when it contains an end-of-file (a `LongInt` value) or version number (a `SignedByte` value).

`ioBuffer`
: A pointer to a data buffer into which data is written by `_Read` calls and from which data is read by `_Write` calls.

`ioReqCount`
: The requested number of bytes to be read or written.

`ioActCount`
: The number of bytes actually read or written.

`ioPosMode`
: The positioning mode (base location) for setting the mark. Bits 0 and 1 of this field indicate how to position the mark; you can use the constants described in "Position Mode Constants" (page 928) to set or test their value. For the functions which use this parameter block, you must have the `kUseWidePositioning` bit set. See "Large Volume Constants" (page 926) for a description of this and other constants.

  You can also use the constants described in "Cache Constants" (page 889) to indicate whether or not to cache the data.

`ioWPosOffset`
: The wide positioning offset to be used in conjunction with the positioning mode specified in the `ioPosMode` field.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

XVolumeParam

Defines an extended volume information parameter block.

```
struct XVolumeParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    unsigned long ioXVersion;
    short ioVolIndex;
    unsigned long ioVCrDate;
    unsigned long ioVLsMod;
    short ioVAtrb;
    unsigned short ioVNmFls;
    unsigned short ioVBitMap;
    unsigned short ioAllocPtr;
    unsigned short ioVNmAlBlks;
    unsigned long ioVAlBlkSiz;
    unsigned long ioVClpSiz;
    unsigned short ioAlBlSt;
    unsigned long ioVNxtCNID;
    unsigned short ioVFrBlk;
    unsigned short ioVSigWord;
    short ioVDrvInfo;
    short ioVDRefNum;
    short ioVFSID;
    unsigned long ioVBkUp;
    short ioVSeqNum;
    unsigned long ioVWrCnt;
    unsigned long ioVFilCnt;
    unsigned long ioVDirCnt;
    long ioVFndrInfo[8];
    UInt64 ioVTotalBytes;
    UInt64 ioVFreeBytes;
};
typedef struct XVolumeParam XVolumeParam;
typedef XVolumeParam * XVolumeParamPtr;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`

The queue type. This field is used internally by the File Manager.

`ioTrap`

The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`

The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`

A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 794) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`ioXVersion`
: The version of the `XVolumeParam` parameter block; currently, this is 0.

`ioVolIndex`
: A volume index for use with the `PBXGetVolInfoSync` (page 782) and `PBXGetVolInfoAsync` (page 779) functions.

`ioVCrDate`
: The date and time that the volume was created (initialized).

`ioVLsMod`
: The date and time that the volume information was last modified. This field is not changed when information is written to a file and does not necessarily indicate when the volume was flushed.

`ioVAtrb`
: The volume attributes. See "Volume Information Attribute Constants" (page 937) for the meanings of the bits in this field.

`ioVNmFls`
: The number of files in the root directory.

`ioVBitMap`
: The first block of the volume bitmap.

`ioAllocPtr`
: The block at which the next new file starts. Used internally.

`ioVNmAlBlks`
: The number of allocation blocks.

`ioVAlBlkSiz`
: The size of the allocation blocks.

`ioVClpSiz`
: The clump size.

`ioAlBlSt`
: The first block in the volume map.

`ioVNxtCNID`
: The next unused catalog node ID.

`ioVFrBlk`
: The number of unused allocation blocks.

`ioVSigWord`
: A signature word identifying the type of volume it's $D2D7 for MFS volumes and $4244 for volumes that support HFS calls.

`ioVDrvInfo`
The drive number of the drive containing the volume.

`ioVDRefNum`
For online volumes, the reference number of the I/O driver for the drive identified by the `ioVDrvInfo` field.

`ioVFSID`
The file-system identifier. It indicates which file system is servicing the volume it's zero for File Manager volumes and nonzero for volumes handled by an external file system.

`ioVBkUp`
The date and time that the volume was last backed up; this is 0 if the volume has never been backed up.

`ioVSeqNum`
Used internally.

`ioVWrCnt`
The volume write count.

`ioVFilCnt`
The total number of files on the volume.

`ioVDirCnt`
The total number of directories (not including the root directory) on the volume.

`ioVFndrInfo`
Information used by the Finder.

`ioVTotalBytes`
The total number of bytes on the volume.

`ioVFreeBytes`
The number of free bytes on the volume.

Discussion
The functions `PBXGetVolInfoSync` and `PBXGetVolInfoAsync` use this parameter block structure to pass arguments and return values.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

Constants

AFP Tag Length Constants

Specify the length of tagged address information for AppleShare volumes.

```
enum {
    kAFPTagLengthIP = 0x06,
    kAFPTagLengthIPPort = 0x08,
    kAFPTagLengthDDP = 0x06
};
```

Constants

`kAFPTagLengthIP`
The length of a 4 byte IP address.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kAFPTagLengthIPPort`
The length of a 4 byte IP address and a 2 byte port.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kAFPTagLengthDDP`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Discussion
These constants are used in the `fLength` field of the `AFPTagData` (page 797) structure to indicate the length, in bytes, of the tagged address information. This length includes the fLength field itself.

AFP Tag Type Constants

Specify the type of tagged address information for AppleShare clients.

```
enum {
    kAFPTagTypeIP = 0x01,
    kAFPTagTypeIPPort = 0x02,
    kAFPTagTypeDDP = 0x03,
    kAFPTagTypeDNS = 0x04
};
```

Constants

`kAFPTagTypeIP`
A basic 4 byte IP address, most significant byte first.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kAFPTagTypeIPPort`
A 4 byte IP address and a 2 byte port number, most significant byte first.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kAFPTagTypeDDP`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kAFPTagTypeDNS`
The address is a DNS name in address:port format. The total length of the DNS name is variable up to 254 characters.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
These constants are used in the `fType` field of the tagged address structure, `AFPTagData` (page 797), to specify the type of address represented by the structure.

Allocation Flags

Indicate how new space is to be allocated.

```
typedef UInt16 FSAllocationFlags;
enum {
    kFSAllocDefaultFlags = 0x0000,
    kFSAllocAllOrNothingMask = 0x0001,
    kFSAllocContiguousMask = 0x0002,
    kFSAllocNoRoundUpMask = 0x0004,
    kFSAllocReservedMask = 0xFFF8
};
```

Constants
`kFSAllocDefaultFlags`
Allocate as much as possible, not necessarily contiguous.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSAllocAllOrNothingMask`
This bit is set when an allocation must allocate the total requested amount, or else fail with nothing allocated; when this bit is not set, the allocation may complete successfully but allocate less than requested.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSAllocContiguousMask`
This bit is set when an allocation should allocate one contiguous range of space on the volume. If this bit is clear, multiple discontiguous extents may be allocated to fulfill the request.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSAllocNoRoundUpMask`
This bit is set when an allocation should no round up to the clump size. If this bit is clear, then additional space beyond the amount requested may be allocated; this is done by some volume formats (including HFS and HFS Plus) to avoid many small allocation requests. If the bit is set, no additional allocation is done (except where required by the volume format, such as rounding up to a multiple of the allocation block size).

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSAllocReservedMask`
Reserved; set to zero.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
If the `kFSAllocContiguousMask` bit is set, then then any newly allocated space must be in one contiguous extent (preferably contiguous with any space already allocated). If `kFSAllocAllOrNothingMask` is set, then the entire `requestCount` bytes must be allocated for the call to succeed; if not set, as many bytes as possible will be allocated (without error). If `kFSAllocNoRoundUpMask` is set, then no additional space is allocated (such as rounding up to a multiple of a clump size); if clear, the volume format may allocate more space than requested as an attempt to reduce fragmentation.

AppleShare Volume Signature

Defines the volume signature for AppleShare volumes.

```
enum {
    AppleShareMediaType = 'afpm'
};
```

Authentication Method Constants

Define the login methods for remote volumes.

```
enum {
    kNoUserAuthentication = 1,
    kPassword = 2,
    kEncryptPassword = 3,
    kTwoWayEncryptPassword = 6
};
```

Constants
`kNoUserAuthentication`
No password.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kPassword`
8-byte password.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kEncryptPassword`
Encrypted 8-byte password.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kTwoWayEncryptPassword`
Two-way random encryption.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
These constants are used in the `uamType` field of an `AFPVolMountInfo` (page 797) structure and in the `ioObjType` field of the parameter block passed to the `PBHGetLogInInfoSync` and `PBHGetLogInInfoAsync` functions to specify the type of user authentication used by a remote volume.

Cache Constants

Indicate whether or not data should be cached.

```
enum {
    pleaseCacheBit = 4,
    pleaseCacheMask = 0x0010,
    noCacheBit = 5,
    noCacheMask = 0x0020,
    rdVerifyBit = 6,
    rdVerifyMask = 0x0040,
    rdVerify = 64,
    forceReadBit = 6,
    forceReadMask = 0x0040,
    newLineBit = 7,
    newLineMask = 0x0080,
    newLineCharMask = 0xFF00
};
```

Constants

`pleaseCacheBit`
Indicates that the data should be cached.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`pleaseCacheMask`
Requests that the data be cached, if possible. You should cache reads and writes if you read or write the same portion of a file multiple times.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`noCacheBit`
Indicates that data should not be cached.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`noCacheMask`
Requests that the data not be cached, if possible. You should not cache reads and writes if you read or write data from a file only once.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`rdVerifyBit`
Indicates that all reads should come from the source and be verified against the data in memory.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`rdVerifyMask`
Requests that all reads (not writes) come directly from the source and be verified against the data in memory. This flushes the cache and sends all read requests to the data source.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`rdVerify`
This is the old name of `rdVerifyMask`. Both request that all reads come directly from the source of the data and be compared against the data in memory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`forceReadBit`
Indicates that reads should come from the disk.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`forceReadMask`
Forces reads from disk, bypassing all caches. Clients can use this to verify that data is stored correctly on the media (for example, to verify after writing) by reading the data into a different buffer while setting the bit, and then comparing the newly read data with the previously written data.

The `forceReadMask` is the same as the `rdVerifyMask` used in the older APIs. The actual implementation of the `rdVerifyMask` in the older APIs actually caused the "force read" behavior, and only compared the data in partial sectors. `FSReadFork` cleans up this behavior by always letting the client do all of the compares.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`newLineBit`
Indicates that newline mode should be used for reads.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`newLineMask`
Requests that newline mode be used for reads. In newline mode, the read stops when one of the following conditions is met:

- The requested number of bytes have been read.
- The end-of-file is reached.
- The newline character has been read. If the newline character is found, it will be the last character put into the buffer and the number of bytes read will include it.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`newLineCharMask`
Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
For the `FSReadFork` and `FSWriteFork` functions, and their parameter block equivalents, you may add either of the `pleaseCacheMask` or `noCacheMask` constants to one of the "Position Mode Constants" (page 928) to hint whether the data should be cached or not.

The `pleaseCacheBit` and the `noCacheBit` are mutually exclusive and only one should be set at a time. If neither bit is set, the program has indicated that it doesn't care if the data is cached or not.

Catalog Information Bitmap Constants

Specify what file or fork information to get or set.

```
enum {
    kFSCatInfoNone = 0x00000000,
    kFSCatInfoTextEncoding = 0x00000001,
    kFSCatInfoNodeFlags = 0x00000002,
    kFSCatInfoVolume = 0x00000004,
    kFSCatInfoParentDirID = 0x00000008,
    kFSCatInfoNodeID = 0x00000010,
    kFSCatInfoCreateDate = 0x00000020,
    kFSCatInfoContentMod = 0x00000040,
    kFSCatInfoAttrMod = 0x00000080,
    kFSCatInfoAccessDate = 0x00000100,
    kFSCatInfoBackupDate = 0x00000200,
    kFSCatInfoPermissions = 0x00000400,
    kFSCatInfoFinderInfo = 0x00000800,
    kFSCatInfoFinderXInfo = 0x00001000,
    kFSCatInfoValence = 0x00002000,
    kFSCatInfoDataSizes = 0x00004000,
    kFSCatInfoRsrcSizes = 0x00008000,
    kFSCatInfoSharingFlags = 0x00010000,
    kFSCatInfoUserPrivs = 0x00020000,
    kFSCatInfoUserAccess = 0x00080000,
    kFSCatInfoSetOwnership = 0x00100000,
    kFSCatInfoAllDates = 0x000003E0,
    kFSCatInfoGettableInfo = 0x0003FFFF,
    kFSCatInfoSettableInfo = 0x00001FE3,
    kFSCatInfoReserved = 0xFFFC0000
};
```

Constants

`kFSCatInfoNone`

No catalog information.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoTextEncoding`

Retrieve or set the text encoding hint, in the `textEncodingHint` field.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoNodeFlags`

Retrieve or set the catalog node flags. Currently, you can only set bits 0 and 4. See "Catalog Information Node Flags" (page 894) for more information on these flags.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoVolume`
Retrieve the volume reference number of the volume on which the file or directory resides.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSCatInfoParentDirID`
Retrieve the parent directory ID.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSCatInfoNodeID`
Retrieve the file or directory ID.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSCatInfoCreateDate`
Retrieve or set the creation date.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSCatInfoContentMod`
Retrieve or set the date that the resource or data fork was last modified.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSCatInfoAttrMod`
Retrieve or set the date that an attribute or named fork was last modified.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSCatInfoAccessDate`
Retrieve or set the date that the fork or file was last accessed.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSCatInfoBackupDate`
Retrieve or set the date that the fork or file was last backed up.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSCatInfoPermissions`
Retrieve or set the file or fork's permissions.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSCatInfoFinderInfo`
Retrieve or set the file or fork's Finder information.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSCatInfoFinderXInfo`
Retrieve or set the file or fork's extended Finder information.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoValence`
For folders only, retrieve the valence of the folder. For files, this is zero.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoDataSizes`
Retrieve the logical and physical size of the data fork.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoRsrcSizes`
Retrieve the logical and physical size of the resource fork.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoSharingFlags`
Retrieve the fork or file's sharing flags: `kioFlAttribMountedBit`, `kioFlAttribSharePointBit`. See "File Attribute Constants" (page 914) for more information on these bits.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoUserPrivs`
Retrieve the file's user privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoUserAccess`
Available in Mac OS X v10.1 and later.

Declared in `Files.h`.

`kFSCatInfoSetOwnership`
Attempt to set the file's user and group (UID and GID). If the File Manager cannot set the the user or group ID, the call fails. (Mac OS X only).

Available in Mac OS X v10.3 and later.

Declared in `Files.h`.

`kFSCatInfoAllDates`
Retrieve or set all of the date information for the fork or file: creation date, modification dates, access date, backup date, etc.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoGettableInfo`
Retrieve all gettable data.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoSettableInfo`
Set all settable data. This includes the flags, dates, permissions, Finder info, and text encoding hint.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoReserved`
Represents bits that are currently reserved.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
These constants are used in the `FSCatalogInfoBitmap` type to specify what file or fork information to get or set. If used with the `FSGetCatalogInfo` or `FSGetCatalogInfoBulk` functions, these constants tell the File Manager which fields to return information in. If used with the `FSSetCatalogInfo` function, these constants tell the File Manager which fields you've filled out with values that it should use to change the fork or file's catalog information.

Catalog Information Node Flags

Define the values used in the `nodeFlags` field of the `FSCatalogInfo` structure.

```
enum {
    kFSNodeLockedBit = 0,
    kFSNodeLockedMask = 0x0001,
    kFSNodeResOpenBit = 2,
    kFSNodeResOpenMask = 0x0004,
    kFSNodeDataOpenBit = 3,
    kFSNodeDataOpenMask = 0x0008,
    kFSNodeIsDirectoryBit = 4,
    kFSNodeIsDirectoryMask = 0x0010,
    kFSNodeCopyProtectBit = 6,
    kFSNodeCopyProtectMask = 0x0040,
    kFSNodeForkOpenBit = 7,
    kFSNodeForkOpenMask = 0x0080,
    kFSNodeHardLinkBit = 8,
    kFSNodeHardLinkMask = 0x00000100
};
```

Constants
`kFSNodeLockedBit`
Set if the file or directory is locked.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSNodeLockedMask`
Indicates that the file or directory is locked.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSNodeResOpenBit`
Set if the resource fork is open.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSNodeResOpenMask`
Indicates that the resource fork is open.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeDataOpenBit`
Set if the data fork is open.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeDataOpenMask`
Indicates that the data fork is open.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeIsDirectoryBit`
Set if the object is a directory.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeIsDirectoryMask`
Indicates that the object is a directory.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeCopyProtectBit`
Set of the file or directory is copy protected.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeCopyProtectMask`
Indicates that the file or directory is copy protected.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeForkOpenBit`
Set if the file or directory has any open fork.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeForkOpenMask`
Indicates that the file or directory has an open fork of any type.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeHardLinkBit`
Available in Mac OS X v10.2 and later.
Declared in `Files.h`.

`kFSNodeHardLinkMask`
Available in Mac OS X v10.2 and later.
Declared in `Files.h`.

Catalog Information Sharing Flags

Indicate the status of a shared directory.

```
enum {
    kFSNodeInSharedBit = 2,
    kFSNodeInSharedMask = 0x0004,
    kFSNodeIsMountedBit = 3,
    kFSNodeIsMountedMask = 0x0008,
    kFSNodeIsSharePointBit = 5,
    kFSNodeIsSharePointMask = 0x0020
};
```

Constants

`kFSNodeInSharedBit`
Set if a directory is within a share point.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeInSharedMask`
Indicates that the directory is within a share point.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeIsMountedBit`
Set if a directory is a share point currently mounted by some user.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeIsMountedMask`
Indicates that the directory is a share point currently mounted by some user.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeIsSharePointBit`
Set if a directory is a share point (an exported volume).
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeIsSharePointMask`
Indicates that the directory is a share point (an exported volume).
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Discussion

The `FSCatalogInfo` (page 826) structure uses these constants in its `sharingFlags` field.

Catalog Search Bits

Indicate the criteria for a catalog search.

```
enum {
    fsSBPartialNameBit = 0,
    fsSBFullNameBit = 1,
    fsSBFlAttribBit = 2,
    fsSBFlFndrInfoBit = 3,
    fsSBFlLgLenBit = 5,
    fsSBFlPyLenBit = 6,
    fsSBFlRLgLenBit = 7,
    fsSBFlRPyLenBit = 8,
    fsSBFlCrDatBit = 9,
    fsSBFlMdDatBit = 10,
    fsSBFlBkDatBit = 11,
    fsSBFlXFndrInfoBit = 12,
    fsSBFlParIDBit = 13,
    fsSBNegateBit = 14,
    fsSBDrUsrWdsBit = 3,
    fsSBDrNmFlsBit = 4,
    fsSBDrCrDatBit = 9,
    fsSBDrMdDatBit = 10,
    fsSBDrBkDatBit = 11,
    fsSBDrFndrInfoBit = 12,
    fsSBDrParIDBit = 13
};
```

Constants

`fsSBPartialNameBit`
Indicates a search by a substring of the name.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBFullNameBit`
Indicates a search by the full name.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBFlAttribBit`
Indicates a search by the file or directory attributes.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBFlFndrInfoBit`
For files only indicates a search by the file's Finder info.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBFlLgLenBit`
For files only; indicates a search by the logical length of the data fork.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBFlPyLenBit`
For files only; indicates a search by the physical length of the data fork.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBFlRLgLenBit`
For files only; indicates a search for the logical length of the resource fork.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlRPyLenBit`
For files only; indicates a search by the physical length of the resource fork.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlCrDatBit`
For files only indicates a search by the file's creation date.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlMdDatBit`
For files only indicates a search by the date of the file's last modification.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlBkDatBit`
For files only indicates a search by the date of the file's last backup.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlXFndrInfoBit`
For files only indicates a search by the file's extended Finder info.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlParIDBit`
For files only indicates a search by the file's parent ID.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBNegateBit`
Indicates a search for all non-matches. That is, if a file or directory matches one of the other specified criteria, it is not returned; if it does not match any of the specified criteria, it is returned.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrUsrWdsBit`
For directories only indicates a search by the directory's Finder info.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrNmFlsBit`
For directories only; indicates a search by the number of files in the directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrCrDatBit`
For directories only indicates a search by the directory's creation date.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrMdDatBit`
For directories only indicates a search by the date of the directory's last modification.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrBkDatBit`
For directories only indicates a search by the date of the directory's last backup.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrFndrInfoBit`
For directories only indicates a search by the directory's additional Finder info.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrParIDBit`
For directories only indicates a search by the directory's parent ID.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Catalog Search Constants

Specify the which catalog information fields to use as search criteria.

```
enum {
    fsSBNodeID = 0x00008000,
    fsSBAttributeModDate = 0x00010000,
    fsSBAccessDate = 0x00020000,
    fsSBPermissions = 0x00040000,
    fsSBNodeIDBit = 15,
    fsSBAttributeModDateBit = 16,
    fsSBAccessDateBit = 17,
    fsSBPermissionsBit = 18
};
```

Constants

`fsSBNodeID`
Search by a range of catalog node ID.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBAttributeModDate`
Search by a range of attribute (fork) modification date.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBAccessDate`
Search by a range of access date.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBPermissions`
Search by a value or mask of permissions.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBNodeIDBit`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBAttributeModDateBit`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBAccessDateBit`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBPermissionsBit`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Catalog Search Masks

Specify the criteria for a catalog search.

```
enum {
    fsSBPartialName = 1,
    fsSBFullName = 2,
    fsSBFlAttrib = 4,
    fsSBFlFndrInfo = 8,
    fsSBFlLgLen = 32,
    fsSBFlPyLen = 64,
    fsSBFlRLgLen = 128,
    fsSBFlRPyLen = 256,
    fsSBFlCrDat = 512,
    fsSBFlMdDat = 1024,
    fsSBFlBkDat = 2048,
    fsSBFlXFndrInfo = 4096,
    fsSBFlParID = 8192,
    fsSBNegate = 16384,
    fsSBDrUsrWds = 8,
    fsSBDrNmFls = 16,
    fsSBDrCrDat = 512,
    fsSBDrMdDat = 1024,
    fsSBDrBkDat = 2048,
    fsSBDrFndrInfo = 4096,
    fsSBDrParID = 8192
};
```

Constants

`fsSBPartialName`
Search by a substring of the name.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBFullName`
Search by the full name.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBFlAttrib`
Search by the file or directory attributes. You can use the attributes to specify that you are searching for a directory, or for a file or directory that is locked by software.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBFlFndrInfo`
For files only search by the file's Finder info.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBFlLgLen`
For files only; search by the logical length of the data fork.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBFlPyLen`
For files only; search by the physical length of the data fork.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBFlRLgLen`
For files only; search for the logical length of the resource fork.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlRPyLen`
For files only; search by the physical length of the resource fork.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlCrDat`
For files only search by the file's creation date.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlMdDat`
For files only search by the date of the file's last modification.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlBkDat`
For files only search by the date of the file's last backup.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlXFndrInfo`
For files only search by the file's extended Finder info.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlParID`
For files only search by the file's parent ID.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBNegate`
Search for all non-matches. That is, if a file or directory matches one of the other specified criteria, it is not returned; if it does not match any of the specified criteria, it is returned.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrUsrWds`
For directories only search by the directory's Finder info.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrNmFls`
For directories only; search by the number of files in the directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrCrDat`
For directories only search by the directory's creation date.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrMdDat`
For directories only search by the date of the directory's last modification.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrBkDat`
For directories only search by the date of the directory's last backup.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrFndrInfo`
For directories only search by the directory's additional Finder info.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrParID`
For directories only search by the directory's parent ID.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
Use these constants in the `ioSearchBits` field of the `PBCatSearchSync` and `PBCatSearchAsync` functions to specify the criteria for your search.

Extended AFP Volume Mounting Information Flag

Specifies a flag used in the `extendedFlags` field of the `AFPXVolMountInfo` structure.

```
enum {
    kAFPExtendedFlagsAlternateAddressMask = 1
};
```

Constants
`kAFPExtendedFlagsAlternateAddressMask`
Indicates that the `alternateAddressOffset` field in the `AFPXVolMountInfo` record is used.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
See the `AFPXVolMountInfo` (page 799) structure for more information.

Extended Volume Attributes

Describe a volume's extended attributes.

```
enum {
    bIsEjectable = 0,
    bSupportsHFSPlusAPIs = 1,
    bSupportsFSCatalogSearch = 2,
    bSupportsFSExchangeObjects = 3,
    bSupports2TBFiles = 4,
    bSupportsLongNames = 5,
    bSupportsMultiScriptNames = 6,
    bSupportsNamedForks = 7,
    bSupportsSubtreeIterators = 8,
    bL2PCanMapFileBlocks = 9
    bParentModDateChanges = 10,
    bAncestorModDateChanges = 11
    bSupportsSymbolicLinks = 13,
    bIsAutoMounted = 14,
    bAllowCDiDataHandler = 17,
    bSupportsExclusiveLocks = 18
    bSupportsJournaling = 19,
    bNoVolumeSizes = 20,
    bIsCaseSensitive = 22,
    bIsCasePreserving = 23,
    bDoNotDisplay = 24
};
```

Constants

`bIsEjectable`
The volume is in an ejectable disk drive .

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bSupportsHFSPlusAPIs`
The volume supports the HFS Plus APIs directly, i.e., the File Manager does not emulate them.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bSupportsFSCatalogSearch`
The volume supports the `FSCatalogSearch` (page 472) operation.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bSupportsFSExchangeObjects`
The volume supports the `FSExchangeObjects` (page 486) function.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bSupports2TBFiles`
The volume supports 2 terabyte files.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bSupportsLongNames`
The volume supports file, directory, and volume names longer than 31 characters.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bSupportsMultiScriptNames`
The volume supports file, directory, and volume names with characters from multiple script systems.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bSupportsNamedForks`
The volume supports named forks other than the data and resource forks.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bSupportsSubtreeIterators`
The volume supports recursive iterators, not at the volume root.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bL2PCanMapFileBlocks`
The volume supports the `Lg2Phys` SPI correctly.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bParentModDateChanges`
On this volume, changing a file or folder causes its parent's modification date to change.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bAncestorModDateChanges`
On this volume, changing a file or folder causes all ancestor modification dates to change.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bSupportsSymbolicLinks`
The volume supports the creation and use of symbolic links (Mac OS X only).

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bIsAutoMounted`
The volume was mounted automatically (Mac OS X only).

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bAllowCDiDataHandler`
QuickTime's CDi data handler is allowed to examine the volume.

Available in Mac OS X v10.1 and later.

Declared in `Files.h`.

`bSupportsExclusiveLocks`
The volume supports exclusive access to files opened for writing.

Available in Mac OS X v10.2 and later.

Declared in `Files.h`.

`bSupportsJournaling`
: The volume supports journaling. This does not indicate whether journaling is currently enabled on the volume.

  Available in Mac OS X v10.3 and later.

  Declared in `Files.h`.

`bNoVolumeSizes`
: The volume is unable to report volume size or free space.

  Available in Mac OS X v10.3 and later.

  Declared in `Files.h`.

`bIsCaseSensitive`
: The volume is case-sensitive.

  Available in Mac OS X v10.3 and later.

  Declared in `Files.h`.

`bIsCasePreserving`
: The volume is preserves case.

  Available in Mac OS X v10.3 and later.

  Declared in `Files.h`.

`bDoNotDisplay`
: The volume should not be displayed in the user interface.

  Available in Mac OS X v10.3 and later.

  Declared in `Files.h`.

Discussion
The `GetVolParmsInfoBuffer` (page 847) structure uses these constants in its `vMExtendedAttributes` field.

FCB Flags

Specify flags that describe the state of a file.

```
enum {
    kioFCBWriteBit = 8,
    kioFCBWriteMask = 0x0100,
    kioFCBResourceBit = 9,
    kioFCBResourceMask = 0x0200,
    kioFCBWriteLockedBit = 10,
    kioFCBWriteLockedMask = 0x0400,
    kioFCBLargeFileBit = 11,
    kioFCBLargeFileMask = 0x0800,
    kioFCBSharedWriteBit = 12,
    kioFCBSharedWriteMask = 0x1000,
    kioFCBFileLockedBit = 13,
    kioFCBFileLockedMask = 0x2000,
    kioFCBOwnClumpBit = 14,
    kioFCBOwnClumpMask = 0x4000,
    kioFCBModifiedBit = 15,
    kioFCBModifiedMask = 0x8000
};
```

Constants

`kioFCBWriteBit`
Set if data can be written to this file.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBWriteMask`
Tests if data can be written to this file.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBResourceBit`
Set if this FCB describes a resource fork.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBResourceMask`
Tests if this FCB describes a resource fork.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBWriteLockedBit`
Set if this file has a locked byte range.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBWriteLockedMask`
Tests if this file has a locked byte range.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBLargeFileBit`
Set if this file may grow beyond 2GB and the cache uses file blocks, not bytes.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBLargeFileMask`
Tests if this file may grow beyond 2GB and the cache uses file blocks, not bytes.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBSharedWriteBit`
Set if this file has shared write permissions.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBSharedWriteMask`
Tests if this file has shared write permissions.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBFileLockedBit`
Set if this file is locked (write-protected).
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBFileLockedMask`
Tests if this file is locked (write-protected).
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBOwnClumpBit`
Set if this file's clump size is specified in the FCB.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBOwnClumpMask`
Tests if this file's clump size is specified in the FCB.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBModifiedBit`
Set if this file has changed since it was last flushed.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBModifiedMask`
Tests if this file has changed since it was last flushed.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Discussion
These constants are used in the `ioFCBFlags` field of the `FCBPBRec` (page 816) returned by the functions `PBGetFCBInfoSync` and `PBGetFCBInfoAsync`.

File Access Permission Constants

Specify the type of read and write access to a file or fork.

```
enum {
    fsCurPerm = 0x00,
    fsRdPerm = 0x01,
    fsWrPerm = 0x02,
    fsRdWrPerm = 0x03,
    fsRdWrShPerm = 0x04,
    fsRdDenyPerm = 0x10,
    fsWrDenyPerm = 0x20
};
```

Constants

`fsCurPerm`

Requests whatever permissions are currently allowed. If write access in unavailable (because the file is locked or the file is already open with write permission), then read permission is granted. Otherwise read/write permission is granted.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsRdPerm`

Requests permission to read the file.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsWrPerm`

Requests permission to write to the file. If write permission is granted, no other access paths are granted write permission. Note, however, that the File Manager does not support write-only access to a file. Thus, `fsWrPerm` is synonymous with `fsRdWrPerm`.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsRdWrPerm`

Requests exclusive read and write permission. If exclusive read/ write permission is granted, no other users are granted permission to write to the file. Other users may, however, be granted permission to read the file.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsRdWrShPerm`

Requests shared read and write permission. Shared read and write permission allows multiple access paths for reading and writing. This is safe only if there is some way of locking portions of the file before writing to them. On volumes that support range locking, you can use the functions `PBLockRangeSync` and `PBUnlockRangeSync` to lock and unlock ranges of bytes within a file. Applications running in Mac OS X version 10.4 or later should use the functions `FSLockRange` and `FSUnlockRange` for this purpose.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsRdDenyPerm`

Requests that any other paths be prevented from having read access. A path cannot be opened if you request read permission (with the `fsRdPerm` constant) but some other path has requested deny-read access. Similarly, the path cannot be opened if you request deny-read permission, but some other path already has read access. This constant is only supported on volumes which return the `bHasOpenDeny` attribute when you call `FSGetVolumeParms`.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsWrDenyPerm`

Requests that any other paths be prevented from having write access. A path cannot be opened if you request write permission (with the `fsWrPerm` constant) but some other path has requested deny-write access. Similarly, the path cannot be opened if you request deny-write permission, but some other path already has write access. This constant is only supported on volumes which return the `bHasOpenDeny` attribute when you call `FSGetVolumeParms`.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
Use these constants to request a type of access to a file or fork, or to deny a type of access to a file or fork to other paths that may request access.

Note that it is possible, in Mac OS 8 and 9, to open a file residing on read-only media with write access. In Mac OS X, however, you cannot open a file with write access on read-only media; the attempt to open the file fails with a `wrPermErr` error.

File and Folder Access Privilege Constants

Specify access privileges for files and directories in the `ioACAccess` field of the `AccessParam` data type.

```
enum {
    kioACAccessOwnerBit = 31,
    kioACAccessOwnerMask = 0x80000000,
    kioACAccessBlankAccessBit = 28,
    kioACAccessBlankAccessMask = 0x10000000,
    kioACAccessUserWriteBit = 26,
    kioACAccessUserWriteMask = 0x04000000,
    kioACAccessUserReadBit = 25,
    kioACAccessUserReadMask = 0x02000000,
    kioACAccessUserSearchBit = 24,
    kioACAccessUserSearchMask = 0x01000000,
    kioACAccessEveryoneWriteBit = 18,
    kioACAccessEveryoneWriteMask = 0x00040000,
    kioACAccessEveryoneReadBit = 17,
    kioACAccessEveryoneReadMask = 0x00020000,
    kioACAccessEveryoneSearchBit = 16,
    kioACAccessEveryoneSearchMask = 0x00010000,
    kioACAccessGroupWriteBit = 10,
    kioACAccessGroupWriteMask = 0x00000400,
    kioACAccessGroupReadBit = 9,
    kioACAccessGroupReadMask = 0x00000200,
    kioACAccessGroupSearchBit = 8,
    kioACAccessGroupSearchMask = 0x00000100,
    kioACAccessOwnerWriteBit = 2,
    kioACAccessOwnerWriteMask = 0x00000004,
    kioACAccessOwnerReadBit = 1,
    kioACAccessOwnerReadMask = 0x00000002,
    kioACAccessOwnerSearchBit = 0,
    kioACAccessOwnerSearchMask = 0x00000001,
    kfullPrivileges = 0x00070007,
    kownerPrivileges = 0x00000007
};
```

Constants

`kioACAccessOwnerBit`
Indicates that the user is the owner of the directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessOwnerMask`
The user is the owner of the directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessBlankAccessBit`
Indicates that the directory has blank access privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessBlankAccessMask`
The directory has blank access privileges. A directory with blank access privileges set ignores the other access privilege bits and uses the access privilege bits of its parent directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessUserWriteBit`
Indicates that the user has write privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessUserWriteMask`
The user has write privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessUserReadBit`
Indicates that the user has read privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessUserReadMask`
The user has read privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessUserSearchBit`
Indicates that the user has search privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessUserSearchMask`
The user has search privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessEveryoneWriteBit`
Indicates that everyone has write privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessEveryoneWriteMask`
Everyone has write privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessEveryoneReadBit`
Indicates that everyone has read privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessEveryoneReadMask`
Everyone has read privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessEveryoneSearchBit`
Indicates that everyone has search privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessEveryoneSearchMask`
Everyone has search privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessGroupWriteBit`
Indicates that the group has write privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessGroupWriteMask`
The group has write privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessGroupReadBit`
Indicates that the group has read privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessGroupReadMask`
The group has read privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessGroupSearchBit`
Indicates that the group has search privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessGroupSearchMask`
The group has search privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessOwnerWriteBit`
Indicates that the owner has write privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessOwnerWriteMask`
The owner has write privileges.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioACAccessOwnerReadBit`
Indicates that the owner has read privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessOwnerReadMask`
The owner has read privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessOwnerSearchBit`
Indicates that the owner has search privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessOwnerSearchMask`
The owner has search privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kfullPrivileges`
Indicates that everyone, including the owner, have all privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kownerPrivileges`
Indicates that only the owner has all privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
See `AccessParam` (page 795).

File Attribute Constants

Define file and directory attributes returned by the `PBGetCatInfoSync` and `PBGetCatInfoAsync` functions.

```
enum {
    kioFlAttribLockedBit = 0,
    kioFlAttribLockedMask = 0x01,
    kioFlAttribResOpenBit = 2,
    kioFlAttribResOpenMask = 0x04,
    kioFlAttribDataOpenBit = 3,
    kioFlAttribDataOpenMask = 0x08,
    kioFlAttribDirBit = 4,
    kioFlAttribDirMask = 0x10,
    ioDirFlg = 4,
    ioDirMask = 0x10,
    kioFlAttribCopyProtBit = 6,
    kioFlAttribCopyProtMask = 0x40,
    kioFlAttribFileOpenBit = 7,
    kioFlAttribFileOpenMask = 0x80,
    kioFlAttribInSharedBit = 2,
    kioFlAttribInSharedMask = 0x04,
    kioFlAttribMountedBit = 3,
    kioFlAttribMountedMask = 0x08,
    kioFlAttribSharePointBit = 5,
    kioFlAttribSharePointMask = 0x20
};
```

Constants

`kioFlAttribLockedBit`
Indicates that the file or directory is locked. Use the functions `PBHSetFLockSync` and `PBHSetFLockAsync` to lock a file or directory. Use the functions `PBHRstFLockSync` and `PBHRstFLockAsync` to unlock a file or directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribLockedMask`
Tests if the file or directory is locked.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribResOpenBit`
Indicates that the resource fork is open. On Mac OS X, this bit is not set if the resource fork of the file has been opened by a process other than the process making the call to `PBHGetCatInfo` or `PBHGetFInfo`.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribResOpenMask`
Tests if the resource fork is open.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribDataOpenBit`
Indicates that the data fork is open. On Mac OS X, this bit is not set if the data fork of the file has been opened by a process other than the process making the call to `PBHGetCatInfo` or `PBHGetFInfo`.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribDataOpenMask`
Tests if the data fork is open.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribDirBit`
Indicates that this is a directory, not a file. This bit is always clear for files, and is always set for directories.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribDirMask`
Tests if this is a directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`ioDirFlg`
Indicates that this is a directory; this is the old name of the `kioFlAttribDirBit`.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`ioDirMask`
Tests if this is a directory; this is the old name of the `kioFlAttribDirMask`.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribCopyProtBit`
Indicates that the file is "copy-protected" by the AppleShare server.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribCopyProtMask`
Tests if the file is "copy-protected" by the AppleShare server.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribFileOpenBit`
Indicates that the file is open. This bit is set if either the data or the resource fork are open. On Mac OS X, this bit is not set if the file has been opened by a process other than the process making the call to `PBHGetCatInfo` or `PBHGetFInfo`.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribFileOpenMask`
Tests if the file is open. The file is open if either the data or the resource fork are open.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribInSharedBit`
Indicates that the directory is within a shared area of the directory hierarchy.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribInSharedMask`
Tests if the directory is within a shared area of the directory hierarchy.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribMountedBit`
Indicates that the directory is a share point that is mounted by a user.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribMountedMask`
Tests if the directory is a share point that is mounted by a user.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribSharePointBit`
Indicates that the directory is a share point.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribSharePointMask`
Tests if the directory is a share point.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
These constants are used in the `ioFlAttrib` fields of the `HFileInfo` (page 849) and `DirInfo` (page 810) structures returned by the functions `PBGetCatInfoSync` and `PBGetCatInfoAsync`.

File Operation Options

Flags you can use to specify how to perform a file operation.

```
enum {
    kFSFileOperationDefaultOptions = 0,
    kFSFileOperationOverwrite = 0x01,
    kFSFileOperationSkipSourcePermissionErrors = 0x02,
    kFSFileOperationDoNotMoveAcrossVolumes = 0x04,
    kFSFileOperationSkipPreflight = 0x08
};
```

Constants
`kFSFileOperationDefaultOptions`
Use the following default options:

- If the destination directory contains an object with the same name as a source object, abort the operation.
- If a source object cannot be read, abort the operation.
- If asked to move an object across volume boundaries, perform the operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSFileOperationOverwrite`
If the destination directory contains an object with the same name as a source object, overwrite the destination object.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSFileOperationSkipSourcePermissionErrors`
If a source object cannot be read, skip the object and continue the operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSFileOperationDoNotMoveAcrossVolumes`
If asked to move an object across volume boundaries, abort the operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSFileOperationSkipPreflight`
Skip the preflight stage for a directory move or copy operation. This option limits the status information that can be returned during the operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

Discussion
These flags may be passed to any of the functions that initiate a file operation. For more information, see "Copying and Moving Objects Using Asynchronous High-Level File Operations" (page 443).

File Operation Stages

Constants used by the File Manager to indicate the current stage of an asynchronous file operation.

```
typedef UInt32 FSFileOperationStage;
enum {
    kFSOperationStageUndefined = 0,
    kFSOperationStagePreflighting = 1,
    kFSOperationStageRunning = 2,
    kFSOperationStageComplete = 3
};
```

Constants
`kFSOperationStageUndefined`
The File Manager has not started the file operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationStagePreflighting`
The File Manager is performing tasks such as calculating the sizes and number of objects in the operation, and checking to make sure there is enough space on the destination volume to complete the operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationStageRunning`
The File Manager is copying or moving the file or directory.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationStageComplete`
The file operation is complete.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

Discussion
These constants are passed back to your file operation status callback function. For more information, see "File Operation Callbacks" (page 788). You can also get the current stage of a file operation by calling a status accessor function such as `FSFileOperationCopyStatus` (page 487).

File Operation Status Dictionary Keys

Keys used to determine the status of a file operation as reported in a status dictionary.

```
const CFStringRef kFSOperationTotalBytesKey;
const CFStringRef kFSOperationBytesCompleteKey;
const CFStringRef kFSOperationBytesRemainingKey;
const CFStringRef kFSOperationTotalObjectsKey;
const CFStringRef kFSOperationObjectsCompleteKey;
const CFStringRef kFSOperationObjectsRemainingKey;
const CFStringRef kFSOperationTotalUserVisibleObjectsKey;
const CFStringRef kFSOperationUserVisibleObjectsCompleteKey;
const CFStringRef kFSOperationUserVisibleObjectsRemainingKey;
const CFStringRef kFSOperationThroughputKey;
```

Constants
`kFSOperationTotalBytesKey`
The value for this key is a CFNumber that represents the total number of bytes that will be moved or copied by this file operation. This value is not available for a directory operation if the `kFSFileOperationSkipPreflight` (page 918) option flag is specified.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationBytesCompleteKey`
The value for this key is a CFNumber that represents the total number of bytes that have already been moved or copied by this file operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationBytesRemainingKey`
The value for this key is a CFNumber that represents the total number of bytes that remain to be moved or copied by this file operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationTotalObjectsKey`
The value for this key is a CFNumber that represents the total number of objects that will be moved or copied by this file operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationObjectsCompleteKey`
The value for this key is a CFNumber that represents the total number of objects that have already been moved or copied by this file operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationObjectsRemainingKey`
The value for this key is a CFNumber that represents the total number of objects that remain to be moved or copied by this file operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationTotalUserVisibleObjectsKey`
The value for this key is a CFNumber that represents the total number of user-visible objects that will be moved or copied by this file operation. In general, an object is user-visible if it is displayed in a Finder window. For example, a package is counted as a single user-visible object even though it typically contains many other objects.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationUserVisibleObjectsCompleteKey`
The value for this key is a CFNumber that represents the total number of user-visible objects that have already been moved or copied by this file operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationUserVisibleObjectsRemainingKey`
The value for this key is a CFNumber that represents the total number of user-visible objects that remain to be moved or copied by this file operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationThroughputKey`
The value for this key is a CFNumber that represents the current throughput of this file operation in bytes per second.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

Discussion
The status dictionary for a file operation is passed back to your status callback function. For more information, see "File Operation Callbacks" (page 788). You can also get the status dictionary for a file operation by calling a status accessor function such as `FSFileOperationCopyStatus` (page 487).

FNMessage

```
typedef UInt32 FNMessage;
enum {
    kFNDirectoryModifiedMessage = 1
};
```

Constants

`kFNDirectoryModifiedMessage`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Foreign Privilege Model Constant

Identifies the A/UX privilege model.

```
enum {
    fsUnixPriv = 1
};
```

Constants

`fsUnixPriv`
Represents a volume that supports the A/UX privilege model.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Discussion

Used in the `vMForeignPrivID` field of the `GetVolParmsInfoBuffer` (page 847).

Group ID Constant

```
enum {
    knoGroup = 0
};
```

Constants

`knoGroup`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Icon Size Constants

Specify the sizes of the desktop database icon types.

```
enum {
    kLargeIconSize = 256,
    kLarge4BitIconSize = 512,
    kLarge8BitIconSize = 1024,
    kSmallIconSize = 64,
    kSmall4BitIconSize = 128,
    kSmall8BitIconSize = 256
};
```

Constants

`kLargeIconSize`
Large black-and-white icon with mask. Corresponding resource type: '`ICN#`'.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kLarge4BitIconSize`
Large 4-bit color icon. Corresponding resource type: '`icl4`'.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kLarge8BitIconSize`
Large 8-bit color icon. Corresponding resource type: '`icl8`'.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kSmallIconSize`
Small black-and-white icon with mask. Corresponding resource type: '`ics#`'.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kSmall4BitIconSize`
Small 4-bit color icon. Corresponding resource type: '`ics4`'.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kSmall8BitIconSize`
Small 8-bit color icon. Corresponding resource type: '`ics8`'.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Discussion

These constants indicate the amount of storage you should allocate for the icon data for each of the icon types specified by the "Icon Type Constants" (page 922). The desktop database functions which set or retrieve icon data–namely, `PBDTAddIconSync`, `PBDTAddIconAsync`, `PBDTGetIconSync`, `PBDTGetIconAsync`, `PBDTGetIconInfoSync`, and `PBDTGetIconInfoAsync` –expect a pointer to the the storage in the `ioDTBuffer` field of the `DTPBRec` (page 813) parameter block and the appropriate constant in the `ioDTReqCount` field.

Icon Type Constants

Specify the icon types for the desktop database.

```
enum {
    kLargeIcon = 1,
    kLarge4BitIcon = 2,
    kLarge8BitIcon = 3,
    kSmallIcon = 4,
    kSmall4BitIcon = 5,
    kSmall8BitIcon = 6,
    kicnsIconFamily = 239
};
```

Constants

`kLargeIcon`
Large black-and-white icon with mask. Corresponding resource type: '`ICN#`'.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Files.h`.

`kLarge4BitIcon`
Large 4-bit color icon. Corresponding resource type: '`icl4`'.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Files.h`.

`kLarge8BitIcon`
Large 8-bit color icon. Corresponding resource type: '`icl8`'.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Files.h`.

`kSmallIcon`
Small black-and-white icon with mask. Corresponding resource type: '`ics#`'.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Files.h`.

`kSmall4BitIcon`
Small 4-bit color icon. Corresponding resource type: '`ics4`'.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Files.h`.

`kSmall8BitIcon`
Small 8-bit color icon. Corresponding resource type: '`ics8`'.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Files.h`.

`kicnsIconFamily`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Files.h`.

Discussion
These constants are used in the `ioIconType` field of the `DTPBRec` (page 813) parameter block.

Invalid Volume Reference Constant

Represents an invalid volume reference number.

```
enum {
    kFSInvalidVolumeRefNum = 0
};
```

Constants
`kFSInvalidVolumeRefNum`
Invalid volume reference number.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Iterator Flags

Indicate whether an iterator iterates over subtrees or just the immediate children of the container.

```
enum {
    kFSIterateFlat = 0,
    kFSIterateSubtree = 1,
    kFSIterateDelete = 2,
    kFSIterateReserved = 0xFFFFFFFC
};
typedef OptionBits FSIteratorFlags;
```

Constants
`kFSIterateFlat`
Iterate over the immediate children of the container only.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSIterateSubtree`
Iterate over the entire subtree rooted at the container.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSIterateDelete`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSIterateReserved`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

kAsyncMountInProgress

```
enum {
    kAsyncMountInProgress = 1,
    kAsyncMountComplete = 2,
    kAsyncUnmountInProgress = 3,
    kAsyncUnmountComplete = 4,
    kAsyncEjectInProgress = 5,
    kAsyncEjectComplete = 6
};
```

Constants

`kAsyncMountInProgress`
Available in Mac OS X v10.2 and later.
Declared in `Files.h`.

`kAsyncMountComplete`
Available in Mac OS X v10.2 and later.
Declared in `Files.h`.

`kAsyncUnmountInProgress`
Available in Mac OS X v10.2 and later.
Declared in `Files.h`.

`kAsyncUnmountComplete`
Available in Mac OS X v10.2 and later.
Declared in `Files.h`.

`kAsyncEjectInProgress`
Available in Mac OS X v10.2 and later.
Declared in `Files.h`.

`kAsyncEjectComplete`
Available in Mac OS X v10.2 and later.
Declared in `Files.h`.

Notification Subscription Options

Options that can be specified at subscription time.

```
enum {
    kFNNoImplicitAllSubscription = (1 << 0),
    kFNNotifyInBackground = (1 << 1)
};
```

Constants

`kFNNoImplicitAllSubscription`
Specify this option if you do not want to receive notifications on this subscription when `FNNotifyAll` is called. By default, any subscription is also implicitly a subscription to wildcard notifications.
Available in Mac OS X v10.1 and later.
Declared in `Files.h`.

`kFNNotifyInBackground`
Specify this option if you want to receive notifications on this subscription when your application is in background. By default, notifications will be coalesced and and delivered when your application becomes foreground.

Available in Mac OS X v10.3 and later.

Declared in `Files.h`.

kHFSCatalogNodeIDsReusedBit

```
enum {
    kHFSCatalogNodeIDsReusedBit = 12,
    kHFSCatalogNodeIDsReusedMask = 1 << kHFSCatalogNodeIDsReusedBit
};
```

Constants

`kHFSCatalogNodeIDsReusedBit`
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `HFSVolumes.h`.

`kHFSCatalogNodeIDsReusedMask`
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `HFSVolumes.h`.

Large Volume Constants

```
enum {
    kWidePosOffsetBit = 8,
    kUseWidePositioning = (1 << kWidePosOffsetBit),
    kMaximumBlocksIn4GB = 0x007FFFFF
};
```

Constants

`kWidePosOffsetBit`
Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kUseWidePositioning`
Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kMaximumBlocksIn4GB`
Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Mapping Code Constants

Specify the type of object to map or return.

```
enum {
    kOwnerID2Name = 1,
    kGroupID2Name = 2,
    kOwnerName2ID = 3,
    kGroupName2ID = 4,
    kReturnNextUser = 1,
    kReturnNextGroup = 2,
    kReturnNextUG = 3
};
```

Constants

`kOwnerID2Name`
Map a user ID to the user name. Used with the `PBHMapIDSync` or `PBHMapIDAsync` functions.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kGroupID2Name`
Map a group ID to the group name. Used with the `PBHMapIDSync` or `PBHMapIDAsync` functions.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kOwnerName2ID`
Map a user name to the user ID. Used with the `PBHMapNameSync` or `PBHMapNameAsync` functions.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kGroupName2ID`
Map a group name to the group ID. Used with the `PBHMapNameSync` or `PBHMapNameAsync` functions.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kReturnNextUser`
Return the next user entry.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kReturnNextGroup`
Return the next group entry.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kReturnNextUG`
Return the next user or group entry.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Discussion

These constants are used in the `ioObjType` field of the `ObjParam` (page 865) parameter block. The first four constants are passed to the `PBHMapIDSync`, `PBHMapIDAsync`, `PBHMapNameSync`, and `PBHMapNameAsync` functions to specify the mapping to be performed. The last three constants are passed to the `PBGetUGEntrySync` or `PBGetUGEntryAsync` functions to specify the type of object to be returned.

Path Conversion Options

Specify how a pathname is converted to an `FSRef` structure by the function `FSPathMakeRefWithOptions` (page 520).

```
enum {
    kFSPathMakeRefDefaultOptions = 0,
    kFSPathMakeRefDoNotFollowLeafSymlink = 0x01
};
```

Constants

`kFSPathMakeRefDefaultOptions`

Use the default options.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSPathMakeRefDoNotFollowLeafSymlink`

When converting a path that refers to a symbolic link, do not follow the link. The new `FSRef` should refer to the link itself.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

Position Mode Constants

Together with an offset, specify a position within a fork.

```
enum {
    fsAtMark = 0,
    fsFromStart = 1,
    fsFromLEOF = 2,
    fsFromMark = 3
};
```

Constants

`fsAtMark`

The starting point is the access path's current position. The offset is ignored.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsFromStart`

The starting point is offset bytes from the start of the fork. The offset must be non-negative.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsFromLEOF`

The starting point is offset bytes from the logical end of the fork. The offset must not be positive.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsFromMark`
The starting point is offset bytes from the access path's current position. The offset may be positive or negative.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
These constants are used in the `ioPosMode` and `positionMode` fields and parameters of the HFS and HFS Plus file access functions. These functions include those for reading from and writing to files or forks, changing the current position within a file or fork, changing the size of a file or fork, and allocating space to a file or fork.

For the `FSReadFork` and `FSWriteFork` calls, you may also add either of the `pleaseCacheMask` or `noCacheMask` constants to hint whether the data should be cached or not. See "Cache Constants" (page 889).

Root Directory Constants

Specify the directory IDs of the root directory of a volume and its parent.

```
enum {
    fsRtParID = 1,
    fsRtDirID = 2
};
```

Constants
`fsRtParID`
Represents the directory ID of the root directory's parent directory. The root directory has no parent this constant is used when specifying the root directory to functions which require the parent directory ID to identify directories.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsRtDirID`
Represents the directory ID of the volume's root directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

User ID Constants

Specify basic user IDs for shared directories.

```
enum {
    knoUser = 0,
    kadministratorUser = 1
};
```

Constants
`knoUser`
Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kadministratorUser`
Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

User Privileges Constants

Specify the user privileges for a directory on a remote volume.

```
enum {
    kioACUserNoSeeFolderBit = 0,
    kioACUserNoSeeFolderMask = 0x01,
    kioACUserNoSeeFilesBit = 1,
    kioACUserNoSeeFilesMask = 0x02,
    kioACUserNoMakeChangesBit = 2,
    kioACUserNoMakeChangesMask = 0x04,
    kioACUserNotOwnerBit = 7,
    kioACUserNotOwnerMask = 0x80
};
```

Constants

`kioACUserNoSeeFolderBit`
Set if the user does not have "See Folders" privileges. Without "See Folders" privileges, the user cannot see other directories in the specified directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACUserNoSeeFolderMask`
Tests if the user has "See Folders" privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACUserNoSeeFilesBit`
Set if the user does not have "See Files" privileges. Without "See Files" privileges, the user cannot open documents or applications in the specified directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACUserNoSeeFilesMask`
Tests if the user has "See Files" privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACUserNoMakeChangesBit`
Set if the user does not have "Make Changes" privileges. Without "Make Changes" privileges, the user cannot create, modify, rename, or delete any file or directory within the specified directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACUserNoMakeChangesMask`
Tests if the user has "Make Changes" privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACUserNotOwnerBit`
: Set if the user is not the owner of the directory.

  Available in Mac OS X v10.0 and later.

  Declared in `Files.h`.

`kioACUserNotOwnerMask`
: Tests whether the user is the owner of the directory.

  Available in Mac OS X v10.0 and later.

  Declared in `Files.h`.

Discussion
These constants are used in the `ioACUser` field of the `HFileInfo` (page 849) and `DirInfo` (page 810) structures returned by the `PBGetCatInfoSync` and `PBGetCatInfoAsync` functions.

Volume Attribute Constants

Bit position constants that specify volume attributes.

```
enum {
    bLimitFCBs = 31,
    bLocalWList = 30,
    bNoMiniFndr = 29,
    bNoVNEdit = 28,
    bNoLclSync = 27,
    bTrshOffLine = 26,
    bNoSwitchTo = 25,
    bNoDeskItems = 20,
    bNoBootBlks = 19,
    bAccessCntl = 18,
    bNoSysDir = 17,
    bHasExtFSVol = 16,
    bHasOpenDeny = 15,
    bHasCopyFile = 14,
    bHasMoveRename = 13,
    bHasDesktopMgr = 12,
    bHasShortName = 11,
    bHasFolderLock = 10,
    bHasPersonalAccessPrivileges = 9,
    bHasUserGroupList = 8,
    bHasCatSearch = 7,
    bHasFileIDs = 6,
    bHasBTreeMgr = 5,
    bHasBlankAccessPrivileges = 4,
    bSupportsAsyncRequests = 3,
    bSupportsTrashVolumeCache = 2
};
enum {
    bHasDirectIO = 1
};
```

Constants

`bLimitFCBs`

The Finder limits the number of file control blocks used during copying to 8 instead of 16.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bLocalWList`

The Finder uses the returned shared volume handle for its local window list.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bNoMiniFndr`

Reserved; always set to 1.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bNoVNEdit`

This volume's name cannot be edited.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bNoLclSync`
Don't let the Finder change the modification date.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bTrshOffLine`
Any time this volume goes offline, it is zoomed to the Trash and unmounted.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bNoSwitchTo`
The Finder will not switch launch to any application on this volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bNoDeskItems`
Don't place objects in this volume on the Finder desktop.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bNoBootBlks`
This volume is not a startup volume. The Startup menu item is disabled. Boot blocks are not copied during copy operations.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bAccessCntl`
This volume supports AppleTalk AFP access-control interfaces. The following functions are supported:

- `PBHGetLogInInfoSync`
- `PBHGetLogInInfoAsync`
- `PBHGetDirAccessSync`
- `PBHGetDirAccessAsync`
- `PBHSetDirAccessSync`
- `PBHSetDirAccessAsync`
- `PBHMapIDSync`
- `PBHMapIDAsync`
- `PBHMapNameSync`
- `PBHMapNameAsync`

Special folder icons are used. The Access Privileges menu command is enabled for disk and folder items. The `ioFlAttrib` field of the parameter block passed to the `PBGetCatInfoSync` and `PBGetCatInfoSync` functions is assumed to be valid.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bNoSysDir`
This volume doesn't support a system directory. Do not switch launch to this volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasExtFSVol`
This volume is an external file system volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasOpenDeny`
This volume supports the `PBHOpenDenySync` , `PBHOpenDenyAsync`, `PBHOpenRFDenySync` and `PBHOpenRFDenyAsync` functions.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasCopyFile`
This volume supports the `PBHCopyFileSync` and `PBHCopyFileAsync` functions, which is used in copy and duplicate operations if both source and destination volumes have the same server address.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasMoveRename`
This volume supports the `PBHMoveRenameSync` and `PBHMoveRenameAsync` functions.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasDesktopMgr`
This volume supports all of the desktop functions.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasShortName`
This volume supports AFP short names.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasFolderLock`
Folders on the volume can be locked, and so they cannot be deleted or renamed.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasPersonalAccessPrivileges`
This volume has local file sharing enabled.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasUserGroupList`
This volume supports the Users and Groups file and thus the AFP privilege functions.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasCatSearch`
: This volume supports the `PBCatSearchSync` and `PBCatSearchAsync` functions.

  Available in Mac OS X v10.0 and later.

  Declared in `Files.h`.

`bHasFileIDs`
: This volume supports the file ID functions, including the `PBExchangeFilesSync` and `PBExchangeFilesAsync` functions.

  Available in Mac OS X v10.0 and later.

  Declared in `Files.h`.

`bHasBTreeMgr`
: Reserved for internal use.

  Available in Mac OS X v10.0 and later.

  Declared in `Files.h`.

`bHasBlankAccessPrivileges`
: This volume supports inherited access privileges for folders (blank access privileges).

  Available in Mac OS X v10.0 and later.

  Declared in `Files.h`.

`bSupportsAsyncRequests`
: This volume correctly handles asynchronous requests at any time.

  Available in Mac OS X v10.0 and later.

  Declared in `Files.h`.

`bSupportsTrashVolumeCache`
: Available in Mac OS X v10.0 and later.

  Declared in `Files.h`.

Discussion
These constants correspond to bit positions in the `vMAttrib` field of the `GetVolParmsInfoBuffer` (page 847) structure returned by the `PBHGetVolParmsSync` (page 695) and `PBHGetVolParmsAsync` (page 694) functions.

Volume Control Block Flags

Used in the `vcbFlags` field of a volume control block to specify information about a volume.

```
enum {
    kVCBFlagsIdleFlushBit = 3,
    kVCBFlagsIdleFlushMask = 0x0008,
    kVCBFlagsHFSPlusAPIsBit = 4,
    kVCBFlagsHFSPlusAPIsMask = 0x0010,
    kVCBFlagsHardwareGoneBit = 5,
    kVCBFlagsHardwareGoneMask = 0x0020,
    kVCBFlagsVolumeDirtyBit = 15,
    kVCBFlagsVolumeDirtyMask = 0x8000
};
```

Constants

`kVCBFlagsIdleFlushBit`
Indicates that the volume should be flushed at idle time.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kVCBFlagsIdleFlushMask`
Flushes the volume at idle time.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kVCBFlagsHFSPlusAPIsBit`
Indicates that the volume directly implements the HFS Plus APIs (rather than emulating them).
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kVCBFlagsHFSPlusAPIsMask`
The volume directly implements the HFS Plus APIs.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kVCBFlagsHardwareGoneBit`
Indicates that the disk driver returned a `hardwareGoneErr` in response to a read or write call.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kVCBFlagsHardwareGoneMask`
Tests if the disk driver returned a `hardwareGoneErr` in response to a read or write call.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kVCBFlagsVolumeDirtyBit`
Indicates that the volume information has changed since the last time the volume was flushed.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kVCBFlagsVolumeDirtyMask`
The volume has changed since the last time the volume was flushed.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Discussion

See `VCB` (page 868) for a description of the volume control block.

Volume Information Attribute Constants

Define volume attributes returned by the functions `PBHGetVInfoSync`, `PBHGetVInfoAsync`, `PBXGetVolInfoSync`, and `PBXGetVolInfoAsync`.

```
enum {
    kioVAtrbDefaultVolumeBit = 5,
    kioVAtrbDefaultVolumeMask = 0x0020,
    kioVAtrbFilesOpenBit = 6,
    kioVAtrbFilesOpenMask = 0x0040,
    kioVAtrbHardwareLockedBit = 7,
    kioVAtrbHardwareLockedMask = 0x0080,
    kioVAtrbSoftwareLockedBit = 15,
    kioVAtrbSoftwareLockedMask = 0x8000
};
```

Constants

`kioVAtrbDefaultVolumeBit`
Indicates that the volume is the default volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioVAtrbDefaultVolumeMask`
Tests if the volume is the default volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioVAtrbFilesOpenBit`
Indicates that there are open files or iterators.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioVAtrbFilesOpenMask`
Tests if there are open files or iterators.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioVAtrbHardwareLockedBit`
Indicates that the volume is locked by a hardware setting. On Mac OS X, the File Manager only sets the software locked bit for CDs and other read-only media; it does not set the hardware locked bit.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioVAtrbHardwareLockedMask`
Tests if the volume is locked by a hardware setting.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioVAtrbSoftwareLockedBit`
Indicates that the volume is locked by software.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioVAtrbSoftwareLockedMask`
Tests if the volume is locked by software.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
These constants are used in the `ioVAtrb` field of the `HVolumeParam` (page 859) parameter block returned by the `PBHGetVInfoSync` (page 690) and `PBHGetVInfoAsync` (page 686) functions, and in the `ioVAtrb` field of the `XVolumeParam` (page 882) parameter block returned by the `PBXGetVolInfoSync` (page 782) and `PBXGetVolInfoAsync` (page 779) functions.

Volume Information Bitmap Constants

Indicate what volume information to set or retrieve.

```
enum {
    kFSVolInfoNone = 0x0000,
    kFSVolInfoCreateDate = 0x0001,
    kFSVolInfoModDate = 0x0002,
    kFSVolInfoBackupDate = 0x0004,
    kFSVolInfoCheckedDate = 0x0008,
    kFSVolInfoFileCount = 0x0010,
    kFSVolInfoDirCount = 0x0020,
    kFSVolInfoSizes = 0x0040,
    kFSVolInfoBlocks = 0x0080,
    kFSVolInfoNextAlloc = 0x0100,
    kFSVolInfoRsrcClump = 0x0200,
    kFSVolInfoDataClump = 0x0400,
    kFSVolInfoNextID = 0x0800,
    kFSVolInfoFinderInfo = 0x1000,
    kFSVolInfoFlags = 0x2000,
    kFSVolInfoFSInfo = 0x4000,
    kFSVolInfoDriveInfo = 0x8000,
    kFSVolInfoGettableInfo = 0xFFFF,
    kFSVolInfoSettableInfo = 0x3004
};
```

Constants
`kFSVolInfoNone`
No volume information.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoCreateDate`
Retrieve the creation date of the volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoModDate`
Retrieve the date of the volume's last modification.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoBackupDate`
Retrieve or set the date of the volume's last backup.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoCheckedDate`
Retrieve the date that the volume was last checked for consistency.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoFileCount`
Retrieve the number of files on the volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoDirCount`
Retrieve the number of directories on the volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoSizes`
Retrieve the total number of bytes on the volume and the number of unused bytes on the volume (in the `totalBytes` and `freeBytes` fields).

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoBlocks`
Retrieve the block information: the block size, the number of total blocks on the volume, and the number of free blocks on the volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoNextAlloc`
Retrieve the address at which to start the next allocation.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoRsrcClump`
Retrieve the resource fork clump size.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoDataClump`
Retrieve the data fork clump size.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoNextID`
Retrieve the next available catalog node ID.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoFinderInfo`
Retrieve or set the volume's Finder information.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoFlags`
Retrieve or set the volume's flags. See "Volume Information Flags" (page 940) for more information on the volume's flags.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoFSInfo`
Retrieve the filesystem ID and signature.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoDriveInfo`
Retrieve the drive information: the drive number and driver reference number.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoGettableInfo`
Retrieve all of the gettable information.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoSettableInfo`
Set all of the settable information. Currently, this is the backup date, Finder information, and flags.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
These constants are used with the `FSVolumeInfoBitmap` (page 845) data type to indicate what volume information to set or retrieve with the functions `FSSetVolumeInfo` (page 543) and `FSGetVolumeInfo` (page 500) , and their corresponding parameter block calls.

Volume Information Flags

Used by the `FSVolumeInfo` structure to specify characteristics of a volume.

```
enum {
    kFSVolFlagDefaultVolumeBit = 5,
    kFSVolFlagDefaultVolumeMask = 0x0020,
    kFSVolFlagFilesOpenBit = 6,
    kFSVolFlagFilesOpenMask = 0x0040,
    kFSVolFlagHardwareLockedBit = 7,
    kFSVolFlagHardwareLockedMask = 0x0080,
    kFSVolFlagSoftwareLockedBit = 15,
    kFSVolFlagSoftwareLockedMask = 0x8000
};
```

Constants

`kFSVolFlagDefaultVolumeBit`
Set if the volume is the default volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolFlagDefaultVolumeMask`
Indicates that the volume is the default volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolFlagFilesOpenBit`
Set if there are open files or iterators.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolFlagFilesOpenMask`
Indicates that there are open files or iterators.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolFlagHardwareLockedBit`
Set if the volume is locked by a hardware setting. On Mac OS X, the File Manager only sets the software locked bit for CDs and other read-only media; it does not set the hardware locked bit.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolFlagHardwareLockedMask`
Indicates that the volume is locked by a hardware setting.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolFlagSoftwareLockedBit`
Set if the volume is locked by software.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolFlagSoftwareLockedMask`
Indicates that the volume is locked by software.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
See the `flags` field of the `FSVolumeInfo` (page 842) structure.

Volume Mount Flags

Define flags used by the volume mounting information structures.

```
enum {
    volMountNoLoginMsgFlagBit = 0,
    volMountNoLoginMsgFlagMask = 0x0001,
    volMountExtendedFlagsBit = 7,
    volMountExtendedFlagsMask = 0x0080,
    volMountInteractBit = 15,
    volMountInteractMask = 0x8000,
    volMountChangedBit = 14,
    volMountChangedMask = 0x4000,
    volMountFSReservedMask = 0x00FF,
    volMountSysReservedMask = 0xFF00
};
```

Constants

`volMountNoLoginMsgFlagBit`
Indicates that any log-in message or greeting dialog will be suppressed.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`volMountNoLoginMsgFlagMask`
Tells the file system to suppress any log-in message or greeting dialog.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`volMountExtendedFlagsBit`
Indicates that the mounting information is a `AFPXVolMountInfo` record for AppleShare Client version 3.7 and later.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`volMountExtendedFlagsMask`
Tells the file system that the mounting information is an `AFPXVolMountInfo` (page 799) record for AppleShare Client version 3.7 and later.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`volMountInteractBit`
Indicates that it's safe for the file system to perform user interaction to mount the volume.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`volMountInteractMask`
Tells the file system that it's safe to perform user interaction to mount the volume.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`volMountChangedBit`
Indicates that the volume was mounted, but the volume mounting information record needs to be updated.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`volMountChangedMask`
Tests if the volume mounting information record needs to be updated.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`volMountFSReservedMask`
Reserved.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`volMountSysReservedMask`
Reserved.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
Bits 0-7 are defined by each file system for its own use; bits 8-15 are reserved for Apple system use. These constants are used in the `flags` fields of the `AFPVolMountInfo` (page 797), `AFPXVolMountInfo` (page 799), and `VolumeMountInfoHeader` (page 873) structures.

Result Codes

The most common result codes returned by File Manager functions are listed below.

Result Code	Value	Description
`dirFulErr`	-33	File directory full. Available in Mac OS X v10.0 and later.
`dskFulErr`	-34	Disk or volume full. Available in Mac OS X v10.0 and later.
`nsvErr`	-35	Volume not found. Available in Mac OS X v10.0 and later.
`ioErr`	-36	I/O error. Available in Mac OS X v10.0 and later.
`bdNamErr`	-37	Bad filename or volume name. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
fnOpnErr	-38	File not open. Available in Mac OS X v10.0 and later.
eofErr	-39	Logical end-of-file reached. Available in Mac OS X v10.0 and later.
posErr	-40	Attempt to position mark before the start of the file. Available in Mac OS X v10.0 and later.
mFulErr	-41	Memory full (open) or file won't fit (load) Available in Mac OS X v10.0 and later.
tmfoErr	-42	Too many files open. Available in Mac OS X v10.0 and later.
fnfErr	-43	File or directory not found; incomplete pathname. Available in Mac OS X v10.0 and later.
wPrErr	-44	Volume is locked through hardware. Available in Mac OS X v10.0 and later.
fLckdErr	-45	File is locked. Available in Mac OS X v10.0 and later.
vLckdErr	-46	Volume is locked through software. Available in Mac OS X v10.0 and later.
fBsyErr	-47	One or more files are open File is busy Directory is not empty. Available in Mac OS X v10.0 and later.
dupFNErr	-48	Duplicate filename and version Destination file already exists File found instead of folder Available in Mac OS X v10.0 and later.
opWrErr	-49	File already open for writing. Available in Mac OS X v10.0 and later.
paramErr	-50	Invalid value passed in a parameter. Your application passed an invalid parameter for dialog options. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
rfNumErr	-51	Invalid reference number. Available in Mac OS X v10.0 and later.
gfpErr	-52	Error during GetFPos, PBGetFPosSync or PBGetFPosAsync. Available in Mac OS X v10.0 and later.
volOffLinErr	-53	Volume is offline. Available in Mac OS X v10.0 and later.
permErr	-54	Attempt to open locked file for writing Permissions error Available in Mac OS X v10.0 and later.
volOnLinErr	-55	Volume already online. Available in Mac OS X v10.0 and later.
nsDrvErr	-56	No such drive. Available in Mac OS X v10.0 and later.
noMacDskErr	-57	Not a Macintosh disk. Available in Mac OS X v10.0 and later.
extFSErr	-58	Volume belongs to an external file system. Available in Mac OS X v10.0 and later.
fsRnErr	-59	Problem during rename. Available in Mac OS X v10.0 and later.
badMDBErr	-60	Bad master directory block. Available in Mac OS X v10.0 and later.
wrPermErr	-61	Read/ write permission doesn't allow writing. Available in Mac OS X v10.0 and later.
lastDskErr	-64	Available in Mac OS X v10.0 and later.
noDriveErr	-64	Drive not installed. Available in Mac OS X v10.0 and later.
firstDskErr	-84	Available in Mac OS X v10.0 and later.
dirNFErr	-120	Directory not found or incomplete pathname. Available in Mac OS X v10.0 and later.
tmwdoErr	-121	Too many working directories open. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
badMovErr	-122	Attempt to move. Available in Mac OS X v10.0 and later.
wrgVolTypErr	-123	Volume does not support Desktop Manager Not an HFS volume Available in Mac OS X v10.0 and later.
volGoneErr	-124	Server volume has been disconnected. Available in Mac OS X v10.0 and later.
fsDSIntErr	-127	non-hardware internal file system error. Available in Mac OS X v10.0 and later.
fsmFFSNotFoundErr	-431	Foreign file system does not exist. Available in Mac OS X v10.0 and later.
fsmBusyFFSErr	-432	File system is busy, cannot be removed. Available in Mac OS X v10.0 and later.
fsmBadFFSNameErr	-433	Name length not 1 <= length <= 31 Available in Mac OS X v10.0 and later.
fsmBadFSDLenErr	-434	FSD size incompatible with current FSM vers Available in Mac OS X v10.0 and later.
fsmDuplicateFSIDErr	-435	FSID already exists. Available in Mac OS X v10.0 and later.
fsmBadFSDVersionErr	-436	FSM version incompatible with FSD Available in Mac OS X v10.0 and later.
fsmNoAlternateStackErr	-437	no alternate stack for HFS CI Available in Mac OS X v10.0 and later.
fsmUnknownFSMMessageErr	-438	unknown message passed to FSM Available in Mac OS X v10.0 and later.
driverHardwareGoneErr	-503	disk driver's hardware was disconnected Available in Mac OS X v10.0 and later.
fidNotFound	-1300	File ID not found Available in Mac OS X v10.0 and later.
fidExists	-1301	File ID already exists Available in Mac OS X v10.0 and later.

Result Code	Value	Description
notAFileErr	-1302	Specified file is a directory Available in Mac OS X v10.0 and later.
diffVolErr	-1303	Files on different volumes Available in Mac OS X v10.0 and later.
catChangedErr	-1304	Catalog has changed and catalog position record may be invalid Available in Mac OS X v10.0 and later.
sameFileErr	-1306	Can't exchange a file with itself Available in Mac OS X v10.0 and later.
badFidErr	-1307	File ID is dangling or doesn't match with the file number Available in Mac OS X v10.0 and later.
notARemountErr	-1308	_Mount allows only remounts and doesn't get one Available in Mac OS X v10.0 and later.
fileBoundsErr	-1309	File's EOF, offset, mark or size is too big Available in Mac OS X v10.0 and later.
fsDataTooBigErr	-1310	File or volume is too big for system Available in Mac OS X v10.0 and later.
volVMBusyErr	-1311	Can't eject because volume is in use by VM Available in Mac OS X v10.0 and later.
badFCBErr	-1327	FCBRecPtr is not valid Available in Mac OS X v10.0 and later.
errFSUnknownCall	-1400	Selector is not recognized by this file system Available in Mac OS X v10.0 and later.
errFSBadFSRef	-1401	An FSRef parameter was invalid. There are several possible causes: The parameter was not optional, but the pointer was NULL. The volume reference number contained within the FSRef does not match a currently mounted volume. This can happen if the volume was unmounted after the FSRef was created. Some other private field inside the FSRef contains a value that could never be valid. If the field value could be valid, but doesn't happen to match the existing volume or in-memory structures, a "not found" error would be returned instead. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errFSBadForkName	-1402	A supplied fork name was invalid (i.e., was syntactically illegal for the given volume). For example, the fork name might contain characters that cannot be stored on the given volume (such as a colon on HFS volumes). Some volume formats do not store fork names in Unicode. These volume formats will attempt to convert the Unicode name to the kind of encoding used by the volume format. If the name could not be converted, errFSBadForkName is returned. Some volume formats only support a limited set of forks, such as the data and resource forks on HFS volumes. For those volumes, if any other fork name is passed, this error is returned. Available in Mac OS X v10.0 and later.
errFSBadBuffer	-1403	A non-optional buffer pointer was NULL , or its size was invalid for the type of data it was expected to contain. In a protected memory system, this could also mean the buffer space is not part of the address space for the calling process. Available in Mac OS X v10.0 and later.
errFSBadForkRef	-1404	A file reference number does not correspond to a fork opened with the FSOpenFork, PBOpenForkSync , or PBOpenForkAsync functions. This could be because that fork has already been closed. Or, you may have passed a reference number created with older APIs (e.g., by the PBHOpenDF functions). A value of zero is never a valid file reference number. Available in Mac OS X v10.0 and later.
errFSBadInfoBitmap	-1405	A FSCatalogInfoBitmap or FSVolumeInfoBitmap has one or more reserved or undefined bits set. This error code can also be returned if a defined bit is set, but the corresponding FSCatalogInfo or FSVolumeInfo field cannot be operated on with that call (for example, trying to use FSSetCatalogInfo to set the valence of a directory). Available in Mac OS X v10.0 and later.
errFSMissingCatInfo	-1406	A FSCatalogInfo pointer is NULL , but is not optional. Or, the FSCatalogInfo is optional and NULL, but the corresponding FSCatalogInfoBitmap is not zero (that is,the bitmap says that one or more of the FSCatalogInfo fields is being passed, but the supplied pointer was NULL). Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errFSNotAFolder	-1407	A parameter was expected to identify a folder, but it identified some other kind of object (e.g., a file) instead. This implies that the specified object exists, but is of the wrong type. For example, one of the parameters to FSCreateFileUnicode is an FSRef of the directory where the file will be created; if the FSRef actually refers to a file, this error is returned. Available in Mac OS X v10.0 and later.
errFSForkNotFound	-1409	An attempt to specify a fork of a given file or directory, but that particular fork does not exist. Available in Mac OS X v10.0 and later.
errFSNameTooLong	-1410	A file or fork name was too long. This means that the given name could never exist; this is different from a "file not found" or errFSForkNotFound error. Available in Mac OS X v10.0 and later.
errFSMissingName	-1411	A required file or fork name parameter was a NULL pointer, or the length of a filename was zero. Available in Mac OS X v10.0 and later.
errFSBadPosMode	-1412	Reserved or invalid bits in a positionMode field were set. For example, the FSReadFork call does not support newline mode, so setting the newline bit or a newline character in the positionMode parameter would cause this error. Available in Mac OS X v10.0 and later.
errFSBadAllocFlags	-1413	Reserved or invalid bits were set in an FSAllocationFlags parameter. Available in Mac OS X v10.0 and later.
errFSNoMoreItems	-1417	There are no more items to return when enumerating a directory or searching a volume. Note that FSCatalogSearch returns this error, whereas the PBCatSearch functions would return eofErr. Available in Mac OS X v10.0 and later.
errFSBadItemCount	-1418	The maximumObjects parameter to FSGetCatalogInfoBulk or FSCatalogSearch was zero. Available in Mac OS X v10.0 and later.
errFSBadSearchParams	-1419	The search criteria to FSCatalogSearch are invalid or inconsistent. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errFSRefsDifferent	-1420	The two FSRef structures passed to FSCompareFSRefs are for different files or directories. Note that a volume format may be able to compare the FSRef structures without searching for the files or directories, so this error may be returned even if one or both of the FSRef structures refers to non-existent objects. Available in Mac OS X v10.0 and later.
errFSForkExists	-1421	An attempt to create a fork, but that fork already exists. Available in Mac OS X v10.0 and later.
errFSBadIteratorFlags	-1422	The flags passed to FSOpenIterator are invalid. Available in Mac OS X v10.0 and later.
errFSIteratorNotFound	-1423	The value of an FSIterator parameter does not correspond to any currently open iterator. Available in Mac OS X v10.0 and later.
errFSIteratorNotSupported	-1424	The iterator flags or container of an FSIterator are not supported by that call. For example, in the initial release, the FSCatalogSearch call only supports an iterator whose container is in the volume's root directory and whose flags are kFSIterateSubtree (i.e., an iterator for the entire volume's contents). Similarly, in the initial release, FSGetCatalogInfoBulk only supports an iterator whose flags are kFSIterateFlat. Available in Mac OS X v10.0 and later.
errFSQuotaExceeded	-1425	The user's quota of disk blocks has been exhausted. Available in Mac OS X v10.2 and later.
afpAccessDenied	-5000	User does not have the correct access to the file Directory cannot be shared Available in Mac OS X v10.0 and later.
afpAuthContinue	-5001	Further information required to complete AFPLogin call. Available in Mac OS X v10.0 and later.
afpBadUAM	-5002	User authentication method is unknown. Available in Mac OS X v10.0 and later.
afpBadVersNum	-5003	Workstation is using an AFP version that the server doesn't recognize. Available in Mac OS X v10.0 and later.
afpBitmapErr	-5004	Bitmap contained bits undefined for call. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
afpCantMove	-5005	Move destination is offspring of source or root was specified. Available in Mac OS X v10.0 and later.
afpDenyConflict	-5006	Requested user permission not possible. Available in Mac OS X v10.0 and later.
afpDirNotEmpty	-5007	Cannot delete non-empty directory. Available in Mac OS X v10.0 and later.
afpDiskFull	-5008	Insufficient free space on volume for operation. Available in Mac OS X v10.0 and later.
afpEofError	-5009	Read beyond logical end-of-file. Available in Mac OS X v10.0 and later.
afpFileBusy	-5010	Cannot delete an open file. Available in Mac OS X v10.0 and later.
afpFlatVol	-5011	Cannot create directory on specified volume. Available in Mac OS X v10.0 and later.
afpItemNotFound	-5012	Unknown user name/ user ID or missing comment / APPL entry. Available in Mac OS X v10.0 and later.
afpLockErr	-5013	Some or all of requested range is locked by another user. Available in Mac OS X v10.0 and later.
afpMiscErr	-5014	Unexpected error encountered during execution. Available in Mac OS X v10.0 and later.
afpNoMoreLocks	-5015	No more ranges can be locked. Available in Mac OS X v10.0 and later.
afpNoServer	-5016	Server is not responding. Available in Mac OS X v10.0 and later.
afpObjectExists	-5017	Specified destination file or directory already exists. Available in Mac OS X v10.0 and later.
afpObjectNotFound	-5018	Specified file or directory does not exist. Available in Mac OS X v10.0 and later.
afpParmErr	-5019	A specified parameter was out of allowable range. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
afpRangeNotLocked	-5020	Specified range was not locked. Available in Mac OS X v10.0 and later.
afpRangeOverlap	-5021	Part of range is already locked. Available in Mac OS X v10.0 and later.
afpSessClosed	-5022	Session closed. Available in Mac OS X v10.0 and later.
afpUserNotAuth	-5023	User authentication failed (usually, password is not correct). Available in Mac OS X v10.0 and later.
afpCallNotSupported	-5024	Unsupported AFP call was made. Available in Mac OS X v10.0 and later.
afpObjectTypeErr	-5025	A directory exists with that name Directory not found Folder locking not supported by volume Object was a file, not a directory Available in Mac OS X v10.0 and later.
afpTooManyFilesOpen	-5026	Maximum open file count reached. Available in Mac OS X v10.0 and later.
afpServerGoingDown	-5027	Server is shutting down. Available in Mac OS X v10.0 and later.
afpCantRename	-5028	AFPRename cannot rename volume. Available in Mac OS X v10.0 and later.
afpDirNotFound	-5029	Unknown directory specified. Available in Mac OS X v10.0 and later.
afpIconTypeError	-5030	Icon size specified is different from existing icon size. Available in Mac OS X v10.0 and later.
afpVolLocked	-5031	Volume is read-only. Available in Mac OS X v10.0 and later.
afpObjectLocked	-5032	Object is M/R/D/W inhibited. Available in Mac OS X v10.0 and later.
afpContainsSharedErr	-5033	The directory contains a share point. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
afpIDNotFound	-5034	File ID not found. Available in Mac OS X v10.0 and later.
afpIDExists	-5035	File ID already exists. Available in Mac OS X v10.0 and later.
afpDiffVolErr	-5036	Available in Mac OS X v10.0 and later.
afpCatalogChanged	-5037	Catalog has changed and search cannot be resumed. Available in Mac OS X v10.0 and later.
afpSameObjectErr	-5038	Source and destination files are the same. Available in Mac OS X v10.0 and later.
afpBadIDErr	-5039	File ID not found. Available in Mac OS X v10.0 and later.
afpPwdSameErr	-5040	Someone tried to change their password to the same password on a mandatory password change. Available in Mac OS X v10.0 and later.
afpPwdTooShortErr	-5041	The password being set is too short: there is a minimum length that must be met or exceeded. Available in Mac OS X v10.0 and later.
afpPwdExpiredErr	-5042	Password has expired on server. Available in Mac OS X v10.0 and later.
afpInsideSharedErr	-5043	The directory is inside a shared directory. Available in Mac OS X v10.0 and later.
afpInsideTrashErr	-5044	The folder being shared is inside the trash folder OR the shared folder is being moved into the trash folder. Available in Mac OS X v10.0 and later.
afpPwdNeedsChangeErr	-5045	The password needs to be changed. Available in Mac OS X v10.0 and later.
afpPwdPolicyErr	-5046	Password does not conform to server's password policy. Available in Mac OS X v10.0 and later.
afpAlreadyLoggedInErr	-5047	User has been authenticated but is already logged in from another machine (and that's not allowed on this server). Available in Mac OS X v10.0 and later.
afpCallNotAllowed	-5048	Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`afpBadDirIDType`	-5060	Not a fixed directory ID volume. Available in Mac OS X v10.0 and later.
`afpCantMountMoreSrvre`	-5061	Maximum number of volumes has been mounted. Available in Mac OS X v10.0 and later.
`afpAlreadyMounted`	-5062	Volume already mounted. Available in Mac OS X v10.0 and later.
`afpSameNodeErr`	-5063	Attempt to log on to a server running on the same machine. Available in Mac OS X v10.0 and later.

Folder Manager Reference

Framework: CoreServices/CoreServices.h

Declared in Folders.h

Overview

The Folder Manager allows you to find and search folders, create new folders, and control how files are routed between folders. Because you can use the Folder Manager to manipulate standard Mac OS folders without relying on their names, your program is tolerant of changes to folder names and easier to localize.

Carbon supports the Folder Manager, although some functions have been deprecated in Mac OS X. You should always check the value of `gestaltFindFolderAttr` in Mac OS X to determine what functionality is available.

Functions by Task

Describing Folders

`GetFolderTypes` (page 970)
Obtains the folder types contained in the global descriptor list.

`IdentifyFolder` (page 971)
Obtains the folder type for the specified folder.

`InvalidateFolderDescriptorCache` (page 972)
Invalidates any prior `FindFolder` results for the specified folder.

`GetFolderName` (page 969) Deprecated in Mac OS X v10.5
Obtains the name of the specified folder.

Manipulating Folders

`FSFindFolder` (page 966)
Obtains location information for system-related directories.

`FindFolder` (page 960)
Obtains location information for system-related directories.

`FindFolderExtended` (page 962) Deprecated in Mac OS X v10.3
Obtains location information for system-related directories. (Deprecated. Use `FindFolder` (page 960) instead.)

`FSFindFolderExtended` (page 966) Deprecated in Mac OS X v10.3
Locates a system-related folder and returns a reference to the folder. (Deprecated. Use `FSFindFolder` (page 966) instead.)

`ReleaseFolder` (page 973) Deprecated in Mac OS X v10.3
Releases the Trash folder in preparation for unmounting a server volume. (Deprecated. This function is not needed in Mac OS X.)

Routing Files

`AddFolderRouting` (page 958) Deprecated in Mac OS X v10.4
Adds a folder routing structure to the global routing list. (Deprecated. There is no replacement function.)

`FindFolderRouting` (page 962) Deprecated in Mac OS X v10.4
Finds the destination folder from a matching folder routing structure for the specified file. (Deprecated. There is no replacement function.)

`GetFolderRoutings` (page 970) Deprecated in Mac OS X v10.4
Obtains folder routing information from the global routing list. (Deprecated. There is no replacement function.)

`RemoveFolderRouting` (page 975) Deprecated in Mac OS X v10.4
Deletes a folder routing structure from the global routing list. (Deprecated. There is no replacement function.)

Working With Folder Manager Notification Functions

`NewFolderManagerNotificationUPP` (page 973)
Creates a new universal procedure pointer (UPP) to a notification function.

`DisposeFolderManagerNotificationUPP` (page 960)
Disposes of the universal procedure pointer (UPP) to a notification function.

`InvokeFolderManagerNotificationUPP` (page 973)
Calls your notification function.

`FolderManagerRegisterCallNotificationProcs` (page 963) Deprecated in Mac OS X v10.3
Calls the registered Folder Manager notification procs. (Deprecated. There is no replacement function.)

`FolderManagerRegisterNotificationProc` (page 964) Deprecated in Mac OS X v10.3
Registers your notification function with the Folder Manager. (Deprecated. There is no replacement function.)

`FolderManagerUnregisterNotificationProc` (page 964) Deprecated in Mac OS X v10.3
Removes your notification function from the Folder Manager's queue. (Deprecated. There is no replacement function.)

Working With Folder Descriptors

`AddFolderDescriptor` (page 957)
Copies the supplied information into a new folder descriptor entry in the system folder list.

`RemoveFolderDescriptor` (page 974)
Deletes the specified folder descriptor entry from the system folder list.

`GetFolderDescriptor` (page 968) Deprecated in Mac OS X v10.3
Obtains the folder descriptor information for the specified folder type from the global descriptor list. (Deprecated. There is no replacement function.)

Finding Files in Special Folders

`FSpDetermineIfSpecIsEnclosedByFolder` (page 967)
Determines whether a file of type `FSSpec` (page 840) is enclosed inside a special folder type for the given domain.

`FSDetermineIfRefIsEnclosedByFolder` (page 965)
Determines whether a file of type `FSRef` (page 837) is enclosed inside a special folder type for the given domain.

`DetermineIfPathIsEnclosedByFolder` (page 959)
Determines whether a file path is enclosed inside a special folder type for the given domain.

Functions

AddFolderDescriptor

Copies the supplied information into a new folder descriptor entry in the system folder list.

```
OSErr AddFolderDescriptor (
   FolderType foldType,
   FolderDescFlags flags,
   FolderClass foldClass,
   FolderLocation foldLocation,
   OSType badgeSignature,
   OSType badgeType,
   ConstStrFileNameParam name,
   Boolean replaceFlag
);
```

Parameters

foldType
Pass a constant identifying the type of the folder you wish the Folder Manager to be able to find. See Folder Type Constants (page 985).

flags
Set these flags to indicate whether a folder is created during startup, if the folder name is locked, and if the folder is created invisible; see Folder Descriptor Flags (page 982).

foldClass
Pass the class of the folder which you wish the Folder Manager to be able to find. The folder class determines how the `foldLocation` parameter is interpreted. See Folder Descriptor Classes (page 981) for a discussion of relative and special folder classes.

foldLocation
For a relative folder, specify the folder type of the parent folder of the target. For a special folder, specify the location of the folder; see Folder Descriptor Locations (page 984).

badgeSignature
Reserved. Pass 0.

badgeType
Reserved. Pass 0.

name
A string specifying the name of the desired folder. For relative folders, this is the exact name of the desired folder. For special folders, the actual target folder may have a different name than the name specified in the folder descriptor. For example, the System Folder is often given a different name, but it can still be located with `FindFolder` (page 960).

replaceFlag
Pass a `Boolean` value indicating whether you wish to replace a folder descriptor that already exists for the specified folder type. If `true`, it replaces the folder descriptor for the specified folder type. If `false`, it does not replace the folder descriptor for the specified folder type.

Return Value
A result code. See "Folder Manager Result Codes" (page 1001). The result code `duplicateFolderDescErr` indicates that a folder descriptor is already installed with the specified folder type and `replaceFlag` is false.

Discussion
The `AddFolderDescriptor` function copies the supplied information into a new descriptor entry in the system folder list. You need to provide folder descriptors for each folder you wish the Folder Manager to be able to find via the function `FindFolder` (page 960). For example, a child folder located in a parent folder needs to have a descriptor created both for it and its parent folder, so that the child can be found. This function is supported under Mac OS 8 and later.

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

Declared In
`Folders.h`

AddFolderRouting

Adds a folder routing structure to the global routing list. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr AddFolderRouting (
   OSType fileType,
   FolderType routeFromFolder,
   FolderType routeToFolder,
   RoutingFlags flags,
   Boolean replaceFlag
);
```

Parameters
fileType
Pass the `OSType` of the file to be routed.

routeFromFolder

Pass the folder type of the "from" folder see Folder Type Constants (page 985) for descriptions of possible values. An item dropped on the folder specified in this parameter will be routed to the folder specified in the `routeToFolder` parameter.

routeToFolder

The folder type of the "to" folder see Folder Type Constants (page 985) for descriptions of possible values.

flags

Reserved for future use; pass 0.

replaceFlag

Pass a `Boolean` value indicating whether you wish to replace a folder routing that already exists. If `true`, it replaces the folder to which the item is being routed. If `false`, it leaves the folder to which the item is being routed.

Return Value

A result code. See "Folder Manager Result Codes" (page 1001). The result code `duplicateRoutingErr` indicates that a folder routing is already installed with the specified folder type and `replaceFlag` is false.

Discussion

Your application can use the `AddFolderRouting` function to specify how the Finder routes a given file 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

`Folders.h`

DetermineIfPathIsEnclosedByFolder

Determines whether a file path is enclosed inside a special folder type for the given domain.

```
OSErr DetermineIfPathIsEnclosedByFolder (
   FSVolumeRefNum domainOrVRefNum,
   OSType folderType,
   const UInt8 *utf8Path,
   Boolean pathIsRealPath,
   Boolean *outResult
);
```

Parameters

domainOrVRefNum

The domain or volume reference number to check. For information about the possible domains, see Disk and Domain Constants (page 997). You can also pass `0` to check all domains and volumes, or you can pass `kOnAppropriateDisk` to check the appropriate volume for the specified file.

folderType

The special folder type to check. For information about the possible folder types, see Folder Type Constants (page 985).

utf8Path

A UTF-8 encoded path to the file for which to search.

pathIsRealPath

A Boolean value that indicates whether the `utf8Path` parameter is guaranteed to be a full and complete path, as opposed to a path containing a symbolic link, an alias, or a relative path.

outResult

A pointer to a Boolean variable. On return, indicates whether or not the file is enclosed inside the special folder type for the given domain.

Discussion
This function provides an efficient way to check to see if a file (or folder) is inside a special folder for a given domain. A typical use for this function is to determine if a given file is inside the trash on a volume:

```
err = DetermineIfPathIsEnclosedByFolder (kOnAppropriateDisk, kTrashFolderType,
 path, false, &result);
```

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

Declared In
`Folders.h`

DisposeFolderManagerNotificationUPP

Disposes of the universal procedure pointer (UPP) to a notification function.

```
void DisposeFolderManagerNotificationUPP (
   FolderManagerNotificationUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

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

Declared In
`Folders.h`

FindFolder

Obtains location information for system-related directories.

```
OSErr FindFolder (
   FSVolumeRefNum vRefNum,
   OSType folderType,
   Boolean createFolder,
   FSVolumeRefNum *foundVRefNum,
   SInt32 *foundDirID
);
```

Parameters

vRefNum

Pass the volume reference number of the volume on which you want to locate a directory, or a constant specifying a disk or domain. The constants which you can use in this parameter are described in Disk and Domain Constants (page 997).

Note that, on Mac OS X, passing a volume reference number in this parameter does not make sense for most of the folder type selectors which you can specify in the `folderType` parameter. On Mac OS X, folders are "domain-oriented"; because there may be more than one domain on any given physical volume, asking for these folders on a per-volume basis yields undefined results. For example, if you were to request the Fonts folder (represented by the selector `kFontsFolderType`) on volume -100, are you requesting the folder /System/Library/Fonts, /Library/Fonts, or ~/Fonts? On Mac OS X you should pass a disk or domain constant in this parameter.

folderType

Pass a four-character folder type, or a constant that represents the type, for the folder you want to find; see Folder Type Constants (page 985).

createFolder

A value of type `Boolean`, as defined in Create Folder Flags (page 981). Pass the constant `kCreateFolder` to create a directory if it does not already exist; otherwise, pass the constant `kDontCreateFolder`. Directories inside the System Folder are created only if the System Folder directory exists. The `FindFolder` function will not create a System Folder directory even if you specify the `kCreateFolder` constant in the `createFolder` parameter. Passing `kCreateFolder` will also not create a parent folder; if the parent of the target folder does not already exist, attempting to create the target will fail.

foundVRefNum

A pointer to a value of type `short`. On return, the value specifies the volume reference number for the volume containing the directory specified in the `folderType` parameter.

foundDirID

A pointer to a value of type `long`. On return, the value specifies the directory ID number for the directory specified in the `folderType` parameter.

Return Value

A result code. See "Folder Manager Result Codes" (page 1001). The result code `fnfErr` indicates that the type has not been found in the `'fld#'` resource, or the disk doesn't have System Folder support, or the disk does not have desktop database support for Desktop Folder—in all cases, the folder has not been found. The result code `dupFNErr` indicates that a file has been found instead of a folder.

Discussion

As of Mac OS 8 and later, your application can add folders to the System Folder—or nest folders within other folders—and locate the folders via the `FindFolder` function. Prior to Mac OS 8, your application could only use `FindFolder` to find folders that were immediately inside of the System Folder, and a few other special folders such as the Trash folder and the System Folder itself. Now, once a folder (and any folders that it is nested within) is described in a folder descriptor—that is, registered using the function `AddFolderDescriptor` (page 957) —your application can use `FindFolder` to find the folder no matter where it is located.

Those folders you're most likely to want to access are Preferences and Trash. For example, you might wish to check for the existence of a user's configuration file in Preferences or, if your application runs out of disk storage when trying to save a file, check how much disk storage is taken by items in the Trash directory and report this to the user.

The specified folder used for a given volume might be located on a different volume; therefore, do not assume the volume that you specify in `vRefNum` and the volume returned through `foundVRefNum` will be the same.

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

Related Sample Code
Simple DrawSprocket

Declared In
`Folders.h`

FindFolderExtended

Obtains location information for system-related directories. (Deprecated in Mac OS X v10.3. Use `FindFolder` (page 960) instead.)

```
OSErr FindFolderExtended (
   FSVolumeRefNum vRefNum,
   OSType folderType,
   Boolean createFolder,
   UInt32 flags,
   void *data,
   FSVolumeRefNum *foundVRefNum,
   SInt32 *foundDirID
);
```

Parameters

foldType

createFolder

flags

Return Value
A result code. See "Folder Manager Result Codes" (page 1001).

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
`Folders.h`

FindFolderRouting

Finds the destination folder from a matching folder routing structure for the specified file. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr FindFolderRouting (
   OSType fileType,
   FolderType routeFromFolder,
   FolderType *routeToFolder,
   RoutingFlags *flags
);
```

Parameters

fileType

Pass the file type specified in the appropriate folder routing structure for the file for which you wish to find a destination folder.

routeFromFolder

Pass the folder type of the "from" folder for which you wish to find a "to" folder see Folder Type Constants (page 985) for descriptions of possible values. An item dropped on the folder specified in this parameter will be routed to the folder specified in the `routeToFolder` parameter.

routeToFolder

A pointer to a value of type `FolderType`. On return, the value is set to the folder type of the destination folder.

flags

Reserved; pass 0.

Return Value
A result code. See "Folder Manager Result Codes" (page 1001).

Discussion
Both the file type and the folder type specified must match those of a folder routing structure in the global routing list for the `FindFolderRouting` function to succeed.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Folders.h`

FolderManagerRegisterCallNotificationProcs

Calls the registered Folder Manager notification procs. (Deprecated in Mac OS X v10.3. There is no replacement function.)

```
OSStatus FolderManagerRegisterCallNotificationProcs (
   OSType message,
   void *arg,
   UInt32 options
);
```

Parameters

message

options

Return Value
A result code. See "Folder Manager Result Codes" (page 1001).

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
`Folders.h`

FolderManagerRegisterNotificationProc

Registers your notification function with the Folder Manager. (Deprecated in Mac OS X v10.3. There is no replacement function.)

```
OSErr FolderManagerRegisterNotificationProc (
   FolderManagerNotificationUPP notificationProc,
   void *refCon,
   UInt32 options
);
```

Parameters

notificationProc
A UPP to your notification function.

refCon
A pointer to client-defined data. This value is passed to your notification function each time it is called.

options
A value specifying registration options. See FolderManagerCallNotificationProcs Options (page 1000).

Return Value
A result code. See "Folder Manager Result Codes" (page 1001).

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
`Folders.h`

FolderManagerUnregisterNotificationProc

Removes your notification function from the Folder Manager's queue. (Deprecated in Mac OS X v10.3. There is no replacement function.)

```
OSErr FolderManagerUnregisterNotificationProc (
   FolderManagerNotificationUPP notificationProc,
   void *refCon
);
```

Parameters

notificationProc
The UPP to your notification function that you passed to the `FolderManagerRegisterNotificationProc` function.

refCon

A pointer to the same value that you passed to the `FolderManagerRegisterNotificationProc` function in the *refCon* parameter.

Return Value
A result code. See "Folder Manager Result Codes" (page 1001).

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
`Folders.h`

FSDetermineIfRefIsEnclosedByFolder

Determines whether a file of type `FSRef` (page 837) is enclosed inside a special folder type for the given domain.

```
OSErr FSDetermineIfRefIsEnclosedByFolder (
   FSVolumeRefNum domainOrVRefNum,
   OSType folderType,
   const FSRef *inRef,
   Boolean *outResult
);
```

Parameters

domainOrVRefNum

The domain or volume reference number to check. For information about the possible domains, see Disk and Domain Constants (page 997). You can also pass `0` to check all domains and volumes, or you can pass `kOnAppropriateDisk` to check the appropriate volume for the specified file.

folderType

The special folder type to check. For information about the possible folder types, see Folder Type Constants (page 985).

inRef

The file for which to search.

outResult

A pointer to a Boolean variable. On return, indicates whether or not the file is enclosed inside the special folder type for the given domain.

Discussion
This function provides an efficient way to check to see if a file (or folder) is inside a special folder for a given domain. A typical use for this function is to determine if a given file is inside the trash on a volume:

```
err = FSDetermineIfRefIsEnclosedByFolder (kOnAppropriateDisk, kTrashFolderType,
 &ref, &result);
```

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

Declared In
`Folders.h`

FSFindFolder

Obtains location information for system-related directories.

```
OSErr FSFindFolder (
   FSVolumeRefNum vRefNum,
   OSType folderType,
   Boolean createFolder,
   FSRef *foundRef
);
```

Parameters

vRefNum

Pass the volume reference number of the volume on which you want to locate a directory, or a constant specifying a disk or domain. The constants which you can use in this parameter are described in Disk and Domain Constants (page 997).

Note that, on Mac OS X, passing a volume reference number in this parameter does not make sense for most of the folder type selectors which you can specify in the `folderType` parameter. On Mac OS X, folders are "domain-oriented"; because there may be more than one domain on any given physical volume, asking for these folders on a per-volume basis yields undefined results. For example, if you were to request the Fonts folder (represented by the selector `kFontsFolderType`)on volume -100, are you requesting the folder /System/Library/Fonts, /Library/Fonts, or ~/Fonts? On Mac OS X you should pass a disk or domain constant in this parameter.

folderType

Pass a four-character folder type, or a constant that represents the type, for the folder you want to find; see Folder Type Constants (page 985).

createFolder

A value of type `Boolean`, as defined in Create Folder Flags (page 981). Pass the constant `kCreateFolder` to create a directory if it does not already exist; otherwise, pass the constant `kDontCreateFolder`. Passing `kCreateFolder` will not create a parent folder; if the parent of the target folder does not already exist, attempting to create the target will fail.

foundRef

A pointer to a file system reference. On return, the `FSRef` refers to the directory specified by the `vRefNum` and `folderType` parameters.

Return Value

A result code. See "Folder Manager Result Codes" (page 1001).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BSDLLCTest

Declared In

`Folders.h`

FSFindFolderExtended

Locates a system-related folder and returns a reference to the folder. (Deprecated in Mac OS X v10.3. Use `FSFindFolder` (page 966) instead.)

```
OSErr FSFindFolderExtended (
   FSVolumeRefNum vRefNum,
   OSType folderType,
   Boolean createFolder,
   UInt32 flags,
   void *data,
   FSRef *foundRef
);
```

Parameters

vRefNum

The volume reference number or domain in which you want to locate a folder. To specify the startup disk, use the constant `kOnSystemDisk`. To specify a domain, use a domain constant such as `kUserDomain`. See Disk and Domain Constants (page 997).

foldType

The type of folder you want to find. See Folder Type Constants (page 985).

createFolder

A value of type `Boolean`, as defined in Create Folder Flags (page 981). Pass the constant `kCreateFolder` to create a folder if it does not already exist; otherwise, pass the constant `kDontCreateFolder`.

flags

An extended behavior constant. See FSFindFolderExtended Flags (page 999).

data

User data which is interpreted differently depending on the constant specified in the `flags` parameter.

foundRef

A pointer to a `FSRef` variable. On return, the variable contains a file system reference to the specified folder.

Return Value
A result code. See "Folder Manager Result Codes" (page 1001).

Discussion
The specified folder might be relocated in future versions of system software; therefore, do not assume the volume that you specify in the `vRefNum` constant and the volume returned in the file system reference will be the same.

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
`Folders.h`

FSpDetermineIfSpecIsEnclosedByFolder

Determines whether a file of type `FSSpec` (page 840) is enclosed inside a special folder type for the given domain. (Deprecated in Mac OS X v10.5.)

```
OSErr FSpDetermineIfSpecIsEnclosedByFolder (
   FSVolumeRefNum domainOrVRefNum,
   OSType folderType,
   const FSSpec *inSpec,
   Boolean *outResult
);
```

Parameters

domainOrVRefNum

The domain or volume reference number to check. For information about the possible domains, see Disk and Domain Constants (page 997). You can also pass `0` to check all domains and volumes, or you can pass `kOnAppropriateDisk` to check the appropriate volume for the specified file.

folderType

The special folder type to check. For information about the possible folder types, see Folder Type Constants (page 985).

inSpec

The file for which to search.

outResult

A pointer to a Boolean variable. On return, indicates whether or not the file is enclosed inside the special folder type for the given domain.

Discussion

This function provides an efficient way to check to see if a file (or folder) is inside a special folder for a given domain. A typical use for this function is to determine if a given file is inside the trash on a volume:

```
err = FSpDetermineIfSpecIsEnclosedByFolder (kOnAppropriateDisk, kTrashFolderType,
 &spec, &result);
```

Availability

Available in Mac OS X v10.4 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Folders.h`

GetFolderDescriptor

Obtains the folder descriptor information for the specified folder type from the global descriptor list. (Deprecated in Mac OS X v10.3. There is no replacement function.)

```
OSErr GetFolderDescriptor (
   FolderType foldType,
   Size descSize,
   FolderDesc *foldDesc
);
```

Parameters

foldType

Pass a constant identifying the type of the folder for which you wish to get descriptor information. See Folder Type Constants (page 985).

descSize

Pass the size (in bytes) of the folder descriptor structure for which a pointer is passed in the `foldDesc` parameter. This value is needed in order to determine the version of the structure being used.

foldDesc

Pass a pointer to a folder descriptor structure. On return, the folder descriptor structure contains information from the global descriptor list for the specified folder type.

Return Value
A result code. See "Folder Manager Result Codes" (page 1001).

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
`Folders.h`

GetFolderName

Obtains the name of the specified folder. (Deprecated in Mac OS X v10.5.)

```
OSErr GetFolderName (
   FSVolumeRefNum vRefNum,
   OSType foldType,
   FSVolumeRefNum *foundVRefNum,
   StrFileName name
);
```

Parameters

vRefNum

Pass the volume reference number (or the constant `kOnSystemDisk` for the startup disk) of the volume containing the folder for which you wish the name to be identified.

foldType

Pass a constant identifying the type of the folder for which you wish the name to be identified. See Folder Type Constants (page 985).

foundVRefNum

On return, a pointer to the volume reference number for the volume containing the folder specified in the `foldType` parameter.

name

On return, a string containing the title of the folder specified in the `foldType` and `vRefNum` parameters.

Return Value
A result code. See "Folder Manager Result Codes" (page 1001).

Discussion
The `GetFolderName` function obtains the name of the folder in the folder descriptor, not the name of the folder on the disk. The names may differ for a few special folders such as the System Folder. For relative folders, however, the actual name is always returned. You typically do not need to call 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
`Folders.h`

GetFolderRoutings

Obtains folder routing information from the global routing list. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr GetFolderRoutings (
   UInt32 requestedRoutingCount,
   UInt32 *totalRoutingCount,
   Size routingSize,
   FolderRouting *theRoutings
);
```

Parameters

requestedRoutingCount
An unsigned 32-bit value. Pass the number of folder routing structures that can fit in the buffer pointed to by the `theRoutings` parameter.

totalRoutingCount
A pointer to an unsigned 32-bit value. On return, the value is set to the number of folder routing structures in the global list. If this value is less than or equal to `requestedRoutingCount`, all folder routing structures were returned to the caller.

routingSize
Pass the size (in bytes) of the `FolderRouting` structure.

theRoutings
Pass a pointer to an array of `FolderRouting` (page 978) structures. On return the structure(s) contain the requested routing information. You may pass `null` if you do not wish this information.

Return Value
A result code. See "Folder Manager Result Codes" (page 1001).

Discussion
The folder routing information in the global routing list determines how the Finder routes 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
`Folders.h`

GetFolderTypes

Obtains the folder types contained in the global descriptor list.

```
OSErr GetFolderTypes (
   UInt32 requestedTypeCount,
   UInt32 *totalTypeCount,
   FolderType *theTypes
);
```

Parameters

requestedTypeCount

Pass the number of `FolderType` values that can fit in the buffer pointed to by the `theTypes` parameter; see Folder Type Constants (page 985).

totalTypeCount

Pass a pointer to an unsigned 32-bit integer value. On return, the value is set to the total number of `FolderType` values in the list. The `totalTypeCount` parameter may produce a value that is larger or smaller than that of the `requestedTypeCount` parameter. If `totalTypeCount` is equal to or smaller than the value passed in for `requestedTypeCount` and the value produced by the `theTypes` parameter is non-null, then all folder types were returned to the caller.

theTypes

Pass a pointer to an array of `FolderType` values; see Folder Type Constants (page 985). On return, the array contains the folder types for the installed descriptors. You can step through the array and call `GetFolderDescriptor` for each folder type. Pass `null` if you only want to know the number of descriptors installed in the system's global list, rather than the actual folder types of those descriptors.

Return Value
A result code. See "Folder Manager Result Codes" (page 1001).

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

Declared In
`Folders.h`

IdentifyFolder

Obtains the folder type for the specified folder.

```
OSErr IdentifyFolder (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   FolderType *foldType
);
```

Parameters

vRefNum

Pass the volume reference number (or the constant `kOnSystemDisk` for the startup disk) of the volume containing the folder whose type you wish to identify.

dirID

Pass the directory ID number for the folder whose type you wish to identify.

foldType

Pass a pointer to a value of type `FolderType`. On return, the value is set to the folder type of the folder with the specified `vRefNum` and `dirID` parameters; see Folder Type Constants (page 985) for descriptions of possible values.

Return Value
A result code. See "Folder Manager Result Codes" (page 1001).

Discussion
The folder type is identified for the folder specified by the `vRefNum` and `dirID` parameters, if such a folder exists. Note that if there are multiple folder descriptors that map to an individual folder, `IdentifyFolder` returns the folder type of only the first matching descriptor that it finds.

Carbon Porting Notes

This function is not useful on Mac OS X.

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

Declared In
`Folders.h`

InvalidateFolderDescriptorCache

Invalidates any prior `FindFolder` results for the specified folder.

```
OSErr InvalidateFolderDescriptorCache (
   FSVolumeRefNum vRefNum,
   SInt32 dirID
);
```

Parameters

vRefNum

Pass the volume reference number (or the constant `kOnSystemDisk` for the startup disk) of the volume containing the folder for which you wish the descriptor cache to be invalidated. Pass 0 to completely invalidate all folder cache information.

dirID

Pass the directory ID number for the folder for which you wish the descriptor cache to be invalidated. Pass 0 to invalidate the cache for all folders on the specified disk.

Return Value
A result code. See "Folder Manager Result Codes" (page 1001).

Discussion
The `InvalidateFolderDescriptorCache` function searches to see if there is currently a cache of results from `FindFolder` calls on the specified folder. If so, it invalidates the cache from the previous calls to the `FindFolder` function in order to force the Folder Manager to reexamine the disk when `FindFolder` is called again on the specified directory ID or volume reference number.

If you remove a directory on disk which you know is a Folder Manager folder, you should call this function to update the Folder Manager.

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

Declared In
`Folders.h`

InvokeFolderManagerNotificationUPP

Calls your notification function.

```
OSStatus InvokeFolderManagerNotificationUPP (
   OSType message,
   void *arg,
   void *userRefCon,
   FolderManagerNotificationUPP userUPP
);
```

Discussion
You should not need to use the `InvokeFolderManagerNotificationUPP` function, as the system calls your notification function for you.

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

Declared In
`Folders.h`

NewFolderManagerNotificationUPP

Creates a new universal procedure pointer (UPP) to a notification function.

```
FolderManagerNotificationUPP NewFolderManagerNotificationUPP (
   FolderManagerNotificationProcPtr userRoutine
);
```

Parameters
`userRoutine`
A pointer to your notification function.

Return Value
The UPP to the notification function.

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

Declared In
`Folders.h`

ReleaseFolder

Releases the Trash folder in preparation for unmounting a server volume. (Deprecated in Mac OS X v10.3. This function is not needed in Mac OS X.)

```
OSErr ReleaseFolder (
   FSVolumeRefNum vRefNum,
   OSType folderType
);
```

Parameters

vRefNum

Pass the volume reference number of the server volume on which you want to release the Trash folder.

folderType

Always pass the `kTrashFolderType` constant. Other folder types are currently ignored.

Return Value
A result code. See "Folder Manager Result Codes" (page 1001).

Discussion
When you call `FindFolder` (page 960) with the `kTrashFolderType` constant, it opens a file on a server volume that ensures each server volume user gets a unique Trash folder. Because a server volume's Trash folder may contain files or folders put there by the user, applications should delete the contents of the server volume's Trash folder. To do this, before your application unmounts a server volume, your application should call `ReleaseFolder`, or the `UnmountVol` request could fail with a `fBsyErr` result code. `ReleaseFolder` closes the file `FindFolder` may have opened and releases the Trash folder on that volume.

Your application should not use this function unless you want to unmount one or more server volumes. Normally, applications should not unmount servers; they should let users use the Finder to unmount volumes. In particular, applications should have no need to release the Trash folder explicitly; rather, unmounting volumes should be left to users to do with the Finder or by restarting.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.

Declared In
`Folders.h`

RemoveFolderDescriptor

Deletes the specified folder descriptor entry from the system folder list.

```
OSErr RemoveFolderDescriptor (
   FolderType foldType
);
```

Parameters

foldType

Pass a constant identifying the type of the folder for which you wish to remove a descriptor. See Folder Type Constants (page 985).

Return Value
A result code. See "Folder Manager Result Codes" (page 1001).

Discussion
Once a folder descriptor has been removed, the function `FindFolder` (page 960) will no longer be able to locate the folder type.

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

Declared In
`Folders.h`

RemoveFolderRouting

Deletes a folder routing structure from the global routing list. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr RemoveFolderRouting (
   OSType fileType,
   FolderType routeFromFolder
);
```

Parameters

`fileType`

Pass the file type value contained in the folder routing structure to be removed.

`routeFromFolder`

Pass the folder type of the "from" folder see Folder Type Constants (page 985) for descriptions of possible values.

Return Value
A result code. See "Folder Manager Result Codes" (page 1001).

Discussion
Both the file type and the folder type specified must match those of an existing folder routing structure for the *`RemoveFolderRouting`* function to succeed.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Folders.h`

Callbacks

FolderManagerNotificationProcPtr

Defines a pointer to a notification function, called for all Folder Manager notifications.

```
typedef OSStatus (*FolderManagerNotificationProcPtr)
(
                OSType message,
                void * arg,
                void * userRefCon);
```

If you name your function `MyFolderManagerNotificationProc`, you would declare it like this:

```
OSStatus MyFolderManagerNotificationProc
(
            OSType message,
            void * arg,
            void * userRefCon);
```

Parameters

message

The type of notification (user login, user logout, etc.). See "Notification Messages" (page 999).

arg

A pointer to additional information, if any. For most messages, this is a pointer to a `FindFolderUserRedirectionGlobals` structure. If the message is `kFolderManagerNotificationDiscardCachedData`, `arg` is undefined.

userRefCon

A pointer to a value for your own use; this may be any value you want, such as a pointer to your globals or other state information.

Return Value

A result code. See "Folder Manager Result Codes" (page 1001).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Folders.h`

Data Types

FindFolderUserRedirectionGlobals

Used in the `arg` parameter of a notification function.

```
struct FindFolderUserRedirectionGlobals {
    UInt32 version;
    UInt32 flags;
    Str31 userName;
    short userNameScript;
    short currentUserFolderVRefNum;
    long currentUserFolderDirID;
    short remoteUserFolderVRefNum;
    long remoteUserFolderDirID;
};
typedef struct FindFolderUserRedirectionGlobals FindFolderUserRedirectionGlobals;
typedef FindFolderUserRedirectionGlobals * FindFolderUserRedirectionGlobalsPtr;
```

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

`Folders.h`

FolderDesc

Used to find existing folder descriptors and create new ones.

```
struct FolderDesc {
    Size descSize;
    FolderType foldType;
    FolderDescFlags flags;
    FolderClass foldClass;
    FolderType foldLocation;
    OSType badgeSignature;
    OSType badgeType;
    UInt32 reserved;
    StrFileName name;
};
typedef struct FolderDesc FolderDesc;
typedef FolderDesc * FolderDescPtr;
```

Fields

`descSize`
The size (in bytes) of this structure.

`foldType`
A constant of type `FolderType` that identifies the kind of target folder. See "Folder Type Constants" (page 985) for a list of possible folder types.

`flags`
Flags indicating whether a folder is created during startup, if the folder name is locked, and if the folder created is invisible; see "Folder Descriptor Flags" (page 982).

`foldClass`
The class indicating whether the folder is relative to the parent folder or special; see "Folder Descriptor Classes" (page 981).

`foldLocation`
For a relative folder, the `foldLocation` field specifies the `FolderType` of the parent folder of the target. For special folders, the location of the folder. See "Folder Descriptor Locations" (page 984).

`badgeSignature`
Reserved. Set this field to 0.

`badgeType`
Reserved. Set this field to 0.

`reserved`
Reserved. Set this field to 0.

`name`
A string specifying the name of the desired folder. For relative folders, this will be the exact name of the desired folder. For special folders, the actual target folder may have a different name than the name specified in the folder descriptor. For example, the System Folder is often given a different name, but it can still be located with `FindFolder` (page 960).

Discussion

The `FolderDesc` structure is supported under Mac OS 8 and later.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Folders.h`

FolderManagerNotificationUPP

Defines a universal procedure pointer (UPP) to a notification function.

```
typedef FolderManagerNotificationProcPtr FolderManagerNotificationUPP;
```

Discussion
For more information, see the description of the `FolderManagerNotificationProcPtr` (page 975) callback function.

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

Declared In
`Folders.h`

FolderRouting

Specifies the folder that files are routed to, based on the folder they are routed from.

```
struct FolderRouting {
    Size descSize;
    OSType fileType;
    FolderType routeFromFolder;
    FolderType routeToFolder;
    RoutingFlags flags;
};
typedef struct FolderRouting FolderRouting;
typedef FolderRouting * FolderRoutingPtr;
```

Fields

`descSize`
The size (in bytes) of this structure.

`fileType`
A constant of type `OSType` that describes the file type of the item to be routed.

`routeFromFolder`
The folder type identifying the folder from which an item will be routed. If an item is dropped on the folder specified in the `routeFromFolder` field, it will be routed to the folder described in the `routeToFolder` field. See "Folder Type Constants" (page 985) for a list of possible values.

`routeToFolder`
The folder type identifying the folder to which an item will be routed; see "Folder Type Constants" (page 985) for a list of possible values.

`flags`
Reserved. Set this field to 0.

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

Declared In
`Folders.h`

MultiUserGestalt

```
struct MultiUserGestalt {
    short giVersion;
    short giReserved0;
    short giReserved1;
    short giReserved2;
    short giReserved3;
    FSSpec giReserved4;
    short giDocsVRefNum;
    long giDocsDirID;
    short giForceSaves;
    short giForceOpens;
    Str31 giSetupName;
    Str31 giUserName;
    Str31 giFrontAppName;
    short giReserved5;
    short giIsOn;
    short giUserLoggedInType;
    char giUserEncryptPwd[16];
    short giUserEnvironment;
    long giReserved6;
    long giReserved7;
    Boolean giDisableScrnShots;
    Boolean giSupportsAsyncFSCalls;
    short giPrefsVRefNum;
    long giPrefsDirID;
    unsigned long giUserLogInTime;
    Boolean giUsingPrintQuotas;
    Boolean giUsingDiskQuotas;
    Boolean giInSystemAccess;
    Boolean giUserFolderEnabled;
    short giReserved8;
    long giReserved9;
    Boolean giInLoginScreen;
};
typedef struct MultiUserGestalt MultiUserGestalt;
typedef MultiUserGestalt * MultiUserGestaltPtr;
```

Fields

`giVersion`
The structure version. A structure version of 0 is invalid.

`giReserved0`
Obsolete with structure version 3.

`giReserved1`
Obsolete.

`giReserved2`
Obsolete with structure version 6.]

`giReserved3`
Obsolete.

`giReserved4`
Obsolete with structure version 6.

`giDocsVRefNum`
The volume reference number associated with the user's documents location.

`giDocsDirID`
The directory ID of the user's documents folder.

`giForceSaves`
`True` if the user is forced to save to their documents folder.

`giForceOpens`
`True` if the user is forced to open from their documents folder.

`giSetupName`
The name of the current setup.

`giUserName`
The name of the current user.

`giFrontAppName`
The name of the frontmost application.

`giReserved5`
Obsolete with structure version 6.

`giIsOn`
`True` if Multiple Users or Macintosh Manager is currently on.

`giUserLoggedInType`
The logged in user type. Zero indicates a normal user, 1 indicates a workgroup administrator, and 2 indicates a global administrator.

`giUserEncryptPwd`
The encrypted user password.

`giUserEnvironment`
The environment that the user has logged into.

`giReserved6`
Obsolete.

`giReserved7`
Obsolete.

`giDisableScrnShots`
`True` if screen shots are not allowed.

`giSupportsAsyncFSCalls`
The Finder uses this to tell if our patches support asynchronous trap patches.

`giPrefsVRefNum`
The volume reference number of preferences.

`giPrefsDirID`
The directory ID of the At Ease Items folder on the preferences volume.

`giUserLogInTime`
The time in seconds the user has been logged in.

`giUsingPrintQuotas`
`True` if the logged in user is using printer quotas.

`giUsingDiskQuotas`
`True` if the logged in user has disk quotas active.

`giInSystemAccess`
`True` if the system is in System Access, that is the owner is logged in.

`giUserFolderEnabled`
`True` if `FindFolder` is redirecting folders.

`giReserved8`
`giReserved9`
`giInLoginScreen`
`True` if no user has logged in, including the owner.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`Folders.h`

Constants

Create Folder Flags

Indicate whether a folder should be created, if it is not found.

```
enum {
    kCreateFolder = true,
    kDontCreateFolder = false
};
```

Constants
`kCreateFolder`
Specifies that the folder should be created, if it is not found.
Available in Mac OS X v10.0 and later.
Declared in `Folders.h`.

`kDontCreateFolder`
Specifies that the folder should not be created, if it is not found.
Available in Mac OS X v10.0 and later.
Declared in `Folders.h`.

Discussion
You can pass these flag constants in the `createFolder` parameter of the function `FSFindFolder` (page 966).

Folder Descriptor Classes

Specify how folder location information should be interpreted.

```
enum {
    kRelativeFolder = 'relf',
    kSpecialFolder = 'spcf'
};
typedef OSType FolderClass;
```

Constants

`kRelativeFolder`

Relative folders are located in terms of the folders in which they are nested, that is, their parent folders. This constant indicates that the folder location specified is the folder type of the parent folder, and the name specified is the name of the folder. Most folder descriptors are for relative folders.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kSpecialFolder`

Special folders—such as the System Folder and the disk's root directory—are in set locations that are not determined relative to any other folder. This constant indicates that the folder is located algorithmically, according to the constant supplied for the folder location (`kBlessedFolder` or `kRootFolder`). Developers cannot create new folder descriptors of the `kSpecialFolder` class.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

Discussion

Constants of type `FolderClass` are used to specify how folder location information should be interpreted in the function `AddFolderDescriptor` (page 957) and the structure `FolderDesc` (page 977). The `FolderClass` constants are supported under Mac OS 8 and later.

Developers can only create new folder descriptors with a class of `kRelativeFolder`.

Folder Descriptor Flags

Specify various attributes of a folder.

```
enum {
    kCreateFolderAtBoot = 0x00000002,
    kCreateFolderAtBootBit = 1,
    kFolderCreatedInvisible = 0x00000004,
    kFolderCreatedInvisibleBit = 2,
    kFolderCreatedNameLocked = 0x00000008,
    kFolderCreatedNameLockedBit = 3,
    kFolderCreatedAdminPrivs = 0x00000010,
    kFolderCreatedAdminPrivsBit = 4
};enum {
    kFolderInUserFolder = 0x00000020,
    kFolderInUserFolderBit = 5,
    kFolderTrackedByAlias = 0x00000040,
    kFolderTrackedByAliasBit = 6,
    kFolderInRemoteUserFolderIfAvailable = 0x00000080,
    kFolderInRemoteUserFolderIfAvailableBit = 7,
    kFolderNeverMatchedInIdentifyFolder = 0x00000100,
    kFolderNeverMatchedInIdentifyFolderBit = 8,
    kFolderMustStayOnSameVolume = 0x00000200,
    kFolderMustStayOnSameVolumeBit = 9,
    kFolderManagerFolderInMacOS9FolderIfMacOSXIsInstalledMask =
0x00000400,
    kFolderManagerFolderInMacOS9FolderIfMacOSXIsInstalledBit = 10,
    kFolderInLocalOrRemoteUserFolder = kFolderInUserFolder |
kFolderInRemoteUserFolderIfAvailable
};
typedef UInt32 FolderDescFlags;
```

Constants

`kCreateFolderAtBoot`
If the bit specified by this mask is set, the folder is created during startup if needed.
Available in Mac OS X v10.0 and later.
Declared in `Folders.h`.

`kCreateFolderAtBootBit`
Available in Mac OS X v10.0 and later.
Declared in `Folders.h`.

`kFolderCreatedInvisible`
If the bit specified by this mask is set, the folder created is invisible.
Available in Mac OS X v10.0 and later.
Declared in `Folders.h`.

`kFolderCreatedInvisibleBit`
Available in Mac OS X v10.0 and later.
Declared in `Folders.h`.

`kFolderCreatedNameLocked`
If the bit specified by this mask is set, the name of the folder is locked when the folder is created.
Available in Mac OS X v10.0 and later.
Declared in `Folders.h`.

`kFolderCreatedNameLockedBit`
Available in Mac OS X v10.0 and later.
Declared in `Folders.h`.

`kFolderCreatedAdminPrivs`
Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kFolderCreatedAdminPrivsBit`
Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

Discussion
The `FolderDescFlags` enumeration defines masks your application can use in the `AddFolderDescriptor` (page 957) function and the `FolderDesc` (page 977) structure to specify various attributes of a folder. All other flag bits are reserved for future use. The `FolderDescFlags` constants are supported under Mac OS 8 and later.

Folder Descriptor Locations

Identify special folder locations.

```
enum {
    kBlessedFolder = 'blsf',
    kRootFolder = 'rotf'
};typedef OSType FolderLocation;
```

Constants
`kBlessedFolder`
Indicates that the folder location is the System Folder on the volume.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kRootFolder`
Indicates that the folder location is the root directory of the volume.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

Discussion
There are two special folder locations that you can specify in folder descriptors via the `FolderDesc` (page 977) structure and the `AddFolderDescriptor` (page 957) function. For folders whose class is `kSpecialFolder`, you can use the following constants to specify the location of the folder algorithmically. The `FolderLocation` constants are supported under Mac OS 8 and later.

kCurrentUserFolderLocation

```
enum {
    kCurrentUserFolderLocation = 'cusf'
};
```

Constants
`kCurrentUserFolderLocation`
Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

Folder Type Constants

Specify a type of folder on a particular volume.

```
enum {
    kSystemFolderType = 'macs',
    kDesktopFolderType = 'desk',
    kSystemDesktopFolderType = 'sdsk',
    kTrashFolderType = 'trsh',
    kSystemTrashFolderType = 'strs',
    kWhereToEmptyTrashFolderType = 'empt',
    kPrintMonitorDocsFolderType = 'prnt',
    kStartupFolderType = 'strt',
    kShutdownFolderType = 'shdf',
    kAppleMenuFolderType = 'amnu',
    kControlPanelFolderType = 'ctrl',
    kSystemControlPanelFolderType = 'sctl',
    kExtensionFolderType = 'extn',
    kFontsFolderType = 'font',
    kPreferencesFolderType = 'pref',
    kSystemPreferencesFolderType = 'sprf',
    kTemporaryFolderType = 'temp'
};
enum {
    kExtensionDisabledFolderType = 'extD',
    kControlPanelDisabledFolderType = 'ctrD',
    kSystemExtensionDisabledFolderType = 'macD',
    kStartupItemsDisabledFolderType = 'strD',
    kShutdownItemsDisabledFolderType = 'shdD',
    kApplicationsFolderType = 'apps',
    kDocumentsFolderType = 'docs'
};
enum {
    kVolumeRootFolderType = 'root',
    kChewableItemsFolderType = 'flnt',
    kApplicationSupportFolderType = 'asup',
    kTextEncodingsFolderType = 'ftex',
    kStationeryFolderType = 'odst',
    kOpenDocFolderType = 'odod',
    kOpenDocShellPlugInsFolderType = 'odsp',
    kEditorsFolderType = 'oded',
    kOpenDocEditorsFolderType = 'fodf',
    kOpenDocLibrariesFolderType = 'odlb',
    kGenEditorsFolderType = 'fedi',
    kHelpFolderType = 'fhlp',
    kInternetPlugInFolderType = 'fnet',
    kModemScriptsFolderType = 'fmod',
    kPrinterDescriptionFolderType = 'ppdf',
    kPrinterDriverFolderType = 'fprd',
    kScriptingAdditionsFolderType = 'fscr',
    kSharedLibrariesFolderType = 'flib',
    kVoicesFolderType = 'fvoc',
    kControlStripModulesFolderType = 'sdev',
    kAssistantsFolderType = 'astf',
    kUtilitiesFolderType = 'utif',
    kAppleExtrasFolderType = 'aexf',
    kContextualMenuItemsFolderType = 'cmnu',
    kMacOSReadMesFolderType = 'morf',
    kALMModulesFolderType = 'walk',
    kALMPreferencesFolderType = 'trip',
    kALMLocationsFolderType = 'fall',
    kColorSyncProfilesFolderType = 'prof',
```

```
    kThemesFolderType = 'thme',
    kFavoritesFolderType = 'favs',
    kInternetFolderType = 'intf',
    kAppearanceFolderType = 'appr',
    kSoundSetsFolderType = 'snds',
    kDesktopPicturesFolderType = 'dtpf',
    kInternetSearchSitesFolderType = 'issf',
    kFindSupportFolderType = 'fnds',
    kFindByContentFolderType = 'fbcf',
    kInstallerLogsFolderType = 'ilgf',
    kScriptsFolderType = 'scrf',
    kFolderActionsFolderType = 'fasf',
    kLauncherItemsFolderType = 'laun',
    kRecentApplicationsFolderType = 'rapp',
    kRecentDocumentsFolderType = 'rdoc',
    kRecentServersFolderType = 'rsvr',
    kSpeakableItemsFolderType = 'spki',
    kKeychainFolderType = 'kchn',
    kQuickTimeExtensionsFolderType = 'qtex',
    kDisplayExtensionsFolderType = 'dspl',
    kMultiprocessingFolderType = 'mpxf',
    kPrintingPlugInsFolderType = 'pplg'
};
typedef OSType FolderType;
```

Constants

`kSystemFolderType`
Specifies the System Folder.
Available in Mac OS X v10.0 and later.
Declared in `Folders.h`.

`kDesktopFolderType`
Specifies the Desktop Folder.
Available in Mac OS X v10.0 and later.
Declared in `Folders.h`.

`kSystemDesktopFolderType`
Available in Mac OS X v10.0 and later.
Declared in `Folders.h`.

`kTrashFolderType`
Specifies the single-user Trash folder.
Available in Mac OS X v10.0 and later.
Declared in `Folders.h`.

`kSystemTrashFolderType`
Available in Mac OS X v10.0 and later.
Declared in `Folders.h`.

`kWhereToEmptyTrashFolderType`
Specifies the shared Trash folder on a file server, this indicates the parent directory of all logged-on users' Trash subdirectories.
Available in Mac OS X v10.0 and later.
Declared in `Folders.h`.

`kPrintMonitorDocsFolderType`
Specifies the PrintMonitor Documents folder in the System Folder.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kStartupFolderType`
Specifies the Startup Items folder in the System Folder.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kShutdownFolderType`
Specifies the Shutdown Items folder in the System Folder.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kAppleMenuFolderType`
Specifies the Apple Menu Items folder in the System Folder.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kControlPanelFolderType`
Specifies the Control Panels folder in the System Folder.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kSystemControlPanelFolderType`
Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kExtensionFolderType`
Specifies the Extensions folder in the System Folder.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kFontsFolderType`
Specifies the Fonts folder in the System Folder.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kPreferencesFolderType`
Specifies the Preferences folder in the System Folder.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kSystemPreferencesFolderType`
Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kTemporaryFolderType`
Specifies the Temporary folder. This folder exists as an invisible folder at the volume root.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kExtensionDisabledFolderType`
Specifies the Extensions (Disabled) folder in the System Folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kControlPanelDisabledFolderType`
Specifies the Control Panels (Disabled) folder in the System Folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kSystemExtensionDisabledFolderType`
Specifies the System Extensions (Disabled) folder in the System Folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kStartupItemsDisabledFolderType`
Specifies the Startup Items (Disabled) folder in the System Folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kShutdownItemsDisabledFolderType`
Specifies the Shutdown Items (Disabled) folder in the System Folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kApplicationsFolderType`
Specifies the Applications folder installed at the root level of the volume. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kDocumentsFolderType`
Specifies the Documents folder. This folder is created at the volume root. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kVolumeRootFolderType`
Specifies the root folder of a volume. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kChewableItemsFolderType`
Specifies the invisible folder on the system disk called "Cleanup at Startup" whose contents are deleted when the system is restarted, instead of merely being moved to the Trash. When the `FindFolder` function indicates this folder is available (by returning `noErr`), developers should usually use this folder for their temporary items, in preference to the Temporary Folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kApplicationSupportFolderType`
Specifies the Application Support folder in the System Folder. This folder contains code and data files needed by third-party applications. These files should usually not be written to after they are installed. In general, files deleted from this folder remove functionality from an application, unlike files in the Preferences folder, which should be non-essential. One type of file that could be placed here would be plug-ins that the user might want to maintain separately from any application, such as for an image-processing application that has many "fourth-party" plug-ins that the user might want to upgrade separately from the host application. Another type of file that might belong in this folder would be application-specific data files that are not preferences, such as for a scanner application that needs to read description files for specific scanner models according to which are currently available on the SCSI bus or network. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kTextEncodingsFolderType`
Specifies the Text Encodings folder in the System Folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kStationeryFolderType`
Specifies the OpenDoc stationery folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kOpenDocFolderType`
Specifies the OpenDoc root folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kOpenDocShellPlugInsFolderType`
Specifies the OpenDoc shell plug-ins folder in the OpenDoc folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kEditorsFolderType`
Specifies the OpenDoc editors folder in the Mac OS folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kOpenDocEditorsFolderType`
Specifies the OpenDoc subfolder in the Editors folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kOpenDocLibrariesFolderType`
Specifies the OpenDoc libraries folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kGenEditorsFolderType`
Specifies a general editors folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kHelpFolderType`
Specifies the Help folder in the System Folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kInternetPlugInFolderType`
Specifies the Browser Plug-ins folder in the System Folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kModemScriptsFolderType`
Specifies the Modem Scripts folder in the Extensions folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kPrinterDescriptionFolderType`
Specifies the Printer Descriptions folder in the Extensions folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kPrinterDriverFolderType`
Specifies the printer drivers folder. This constant is not currently supported.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kScriptingAdditionsFolderType`
Specifies the Scripting Additions folder in the System Folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kSharedLibrariesFolderType`
Specifies the general shared libraries folder. This constant is not currently supported.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kVoicesFolderType`
Specifies the Voices folder in the Extensions folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kControlStripModulesFolderType`
Specifies the Control Strip Modules folder in the System Folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kAssistantsFolderType`
Specifies the Assistants folder installed at the root level of the volume. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kUtilitiesFolderType`
Specifies the Utilities folder installed at the root level of the volume. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kAppleExtrasFolderType`
Specifies the Apple Extras folder installed at the root level of the volume. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kContextualMenuItemsFolderType`
Specifies the Contextual Menu Items folder in the System Folder. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kMacOSReadMesFolderType`
Specifies the Mac OS Read Me Files folder installed at the root level of the volume. Supported with Mac OS 8 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kALMModulesFolderType`
Specifies the Location Manager Modules folder in the Extensions Folder. Supported with Mac OS 8.1 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kALMPreferencesFolderType`
Specifies the Location Manager Prefs folder in the Preferences folder. Supported with Mac OS 8.1 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kALMLocationsFolderType`
Specifies the Locations folder in the Location Manager Prefs folder. Files containing configuration information for different locations are stored here. Supported with Mac OS 8.1 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kColorSyncProfilesFolderType`
Specifies the ColorSync Profiles folder in the System Folder. Supported with Mac OS 8.1 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kThemesFolderType`
Specifies the Theme Files folder in the Appearance folder. Supported with Mac OS 8.1 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kFavoritesFolderType`
Specifies the Favorites folder in the System Folder. This folder is for storing Internet location files, aliases, and aliases to other frequently used items. Facilities for adding items into this folder are found in Contextual Menus, the Finder, Navigation Services, and others. Supported with Mac OS 8.1 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kInternetFolderType`
Specifies the Internet folder installed at the root level of the volume. This folder is a location for saving Internet-related applications, resources, and tools. Supported with Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kAppearanceFolderType`
Specifies the Appearance folder in the System Folder. Supported with Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kSoundSetsFolderType`
Specifies the Sound Sets folder in the Appearance folder. Supported with Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kDesktopPicturesFolderType`
Specifies the Desktop Pictures folder in the Appearance folder. This folder is used for storing desktop picture files. Files of type `'JPEG'` are auto-routed into this folder when dropped into the System Folder. Supported with Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kInternetSearchSitesFolderType`
Specifies the Internet Search Sites folder in the System Folder. This folder contains Internet search site specification files used by the Find application when it accesses Internet search sites. Files of type `'issp'` are auto-routed to this folder. Supported with Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kFindSupportFolderType`
Specifies the Find folder in the Extensions folder. This folder contains files used by the Find application. Supported with Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kFindByContentFolderType`
Specifies the Find By Content folder installed at the root level of the volume. This folder is invisible and its use is private to Find By Content. Supported with Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kInstallerLogsFolderType`
Specifies the Installer Logs folder installed at the root level of the volume. You can use this folder to save installer log files. Supported with Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kScriptsFolderType`
Specifies the Scripts folder in the System Folder. This folder is for saving AppleScript scripts. Supported with Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kFolderActionsFolderType`
Specifies the Folder Action Scripts folder in the Scripts folder. Supported with Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kLauncherItemsFolderType`
Specifies the Launcher Items folder in the System Folder. Items in this folder appear in the Launcher control panel. Items included in folders with names beginning with a bullet (Option-8) character will appear as a separate panel in the Launcher window. Supported with Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kRecentApplicationsFolderType`
Specifies the Recent Applications folder in the Apple Menu Items folder. Apple Menu Items saves aliases to recent applications here. Supported with Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kRecentDocumentsFolderType`

Specifies the Recent Documents folder in the Apple Menu Items folder. Apple Menu Items saves aliases to recently opened documents here. Supported with Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kRecentServersFolderType`

Specifies the Recent Servers folder in the Apple Menu Items folder. Apple Menu Items saves aliases to recently mounted servers here. Supported with Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kSpeakableItemsFolderType`

Specifies the Speakable Items folder. This folder is for storing scripts and items recognized by speech recognition. Supported with Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

kDomainTopLevelFolderType

```
enum {
    kDomainTopLevelFolderType = 'dtop',
    kDomainLibraryFolderType = 'dlib',
    kColorSyncFolderType = 'sync',
    kColorSyncCMMFolderType = 'ccmm',
    kColorSyncScriptingFolderType = 'cscr',
    kPrintersFolderType = 'impr',
    kSpeechFolderType = 'spch',
    kCarbonLibraryFolderType = 'carb',
    kDocumentationFolderType = 'info',
    kDeveloperDocsFolderType = 'ddoc',
    kDeveloperHelpFolderType = 'devh',
    kISSDownloadsFolderType = 'issd',
    kUserSpecificTmpFolderType = 'utmp',
    kCachedDataFolderType = 'cach',
    kFrameworksFolderType = 'fram',
    kPrivateFrameworksFolderType = 'pfrm',
    kClassicDesktopFolderType = 'sdsk',
    kDeveloperFolderType = 'devf',
    kSystemSoundsFolderType = 'ssnd',
    kComponentsFolderType = 'cmpd',
    kQuickTimeComponentsFolderType = 'wcmp',
    kCoreServicesFolderType = 'csrv',
    kPictureDocumentsFolderType = 'pdoc',
    kMovieDocumentsFolderType = 'mdoc',
    kMusicDocumentsFolderType = 'doc',
    kInternetSitesFolderType = 'site',
    kPublicFolderType = 'pubb',
    kAudioSupportFolderType = 'adio',
    kAudioSoundsFolderType = 'asnd',
    kAudioSoundBanksFolderType = 'bank',
    kAudioAlertSoundsFolderType = 'alrt',
    kAudioPlugInsFolderType = 'aplg',
    kAudioComponentsFolderType = 'acmp',
    kKernelExtensionsFolderType = 'kext',
    kDirectoryServicesFolderType = 'dsrv',
    kDirectoryServicesPlugInsFolderType = 'dplg',
    kInstallerReceiptsFolderType = 'rcpt',
    kFileSystemSupportFolderType = 'fsys',
    kAppleShareSupportFolderType = 'shar',
    kAppleShareAuthenticationFolderType = 'auth',
    kMIDIDriversFolderType = 'midi',
    kKeyboardLayoutsFolderType = 'klay',
    kIndexFilesFolderType = 'indx',
    kFindByContentIndexesFolderType = 'fbcx',
    kManagedItemsFolderType = 'mang',
    kBootTimeStartupItemsFolderType = 'empz'
};
```

kAppleshareAutomountServerAliasesFolderType

```
enum {
    kAppleshareAutomountServerAliasesFolderType = 'srvƒ',
    kPreMacOS91ApplicationsFolderType = 'åpps',
```

```
    kPreMacOS91InstallerLogsFolderType = 'îlgf',
    kPreMacOS91AssistantsFolderType = 'åstƒ',
    kPreMacOS91UtilitiesFolderType = 'ütiƒ',
    kPreMacOS91AppleExtrasFolderType = 'åexƒ',
    kPreMacOS91MacOSReadMesFolderType = 'orƒ',
    kPreMacOS91InternetFolderType = 'întƒ',
    kPreMacOS91AutomountedServersFolderType = 'ßrvƒ',
    kPreMacOS91StationeryFolderType = 'ødst'
};
```

kUsersFolderType

```
enum {
    kUsersFolderType = 'usrs',
    kCurrentUserFolderType = 'cusr',
    kCurrentUserRemoteFolderLocation = 'rusf',
    kCurrentUserRemoteFolderType = 'rusr',
    kSharedUserDataFolderType = 'sdat',
    kVolumeSettingsFolderType = 'vsfd'
};
```

kLocalesFolderType

```
enum {
    kLocalesFolderType = 'ƒloc',
    kFindByContentPluginsFolderType = 'fbcp'
};
```

Disk and Domain Constants

Identify the disk or domain in which to locate a folder.

```
enum {
    kOnSystemDisk = -32768L,
    kOnAppropriateDisk = -32767,
    kSystemDomain = -32766,
    kLocalDomain = -32765,
    kNetworkDomain = -32764,
    kUserDomain = -32763,
    kClassicDomain = -32762
};
enum {
    kLastDomainConstant = kUserDomain
};
```

Constants

`kOnSystemDisk`

Specifies the system disk.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kOnAppropriateDisk`
In most cases, the equivalent of `kOnSystemDisk`. On Mac OS X, use this constant instead of the constant `kOnSytemDisk` to indicate any disk.

Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kSystemDomain`
Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kLocalDomain`
Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kNetworkDomain`
Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kUserDomain`
Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kClassicDomain`
Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

`kLastDomainConstant`
Available in Mac OS X v10.0 and later.

Declared in `Folders.h`.

Discussion
You can pass this constant in the `vRefNum` parameter of `FSFindFolder` (page 966) to locate a folder on the startup disk.

Notification Options

Specify options for the `FolderManagerRegisterNotificationProc` function.

```
enum {
    kDoNotRemoveWhenCurrentApplicationQuitsBit = 0,
    kDoNotRemoveWheCurrentApplicationQuitsBit =
kDoNotRemoveWhenCurrentApplicationQuitsBit
};
```

Constants
`kDoNotRemoveWhenCurrentApplicationQuitsBit`
Tells the Folder Manager to not remove your notification function when the current application quits. Otherwise, a notification function registered within an application's context will be automatically removed when that application quits. Programs that register notifications at system startup should set this bit.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Folders.h`.

`kDoNotRemoveWheCurrentApplicationQuitsBit`
Use `kDoNotRemoveWhenCurrentApplicationQuitsBit` instead.
Available in Mac OS X v10.0 through Mac OS X v10.4.
Declared in `Folders.h`.

FSFindFolderExtended Flags

Specify additional options for folder searches performed with the `FSFindFolderExtended` function.

```
enum {
    kFindFolderExtendedFlagsDoNotFollowAliasesBit = 0,
    kFindFolderExtendedFlagsDoNotUseUserFolderBit = 1,
    kFindFolderExtendedFlagsUseOtherUserRecord = 0x01000000
};
```

Discussion
These are passed to `FSFindFolderExtended` (page 966) and `FindFolderExtended` (page 962) in the `flags` field.

FindFolderUserRedirectionGlobals Flags

Used in the `flags` field of the `FindFolderUserRedirectionGlobals` structure

```
enum {
    kFindFolderRedirectionFlagUseDistinctUserFoldersBit = 0,
    kFindFolderRedirectionFlagUseGivenVRefAndDirIDAsUserFolderBit
= 1,
    kFindFolderRedirectionFlagsUseGivenVRefNumAndDirIDAsRemoteUserFolderBit
= 2
};
typedef UInt32 RoutingFlags;
```

FindFolderUserRedirectionGlobals Structure Version

Represents the current version of the `FindFolderUserRedirectionGlobals` structure.

```
enum {
    kFolderManagerUserRedirectionGlobalsCurrentVersion = 1
};
```

Notification Messages

Define messages sent to your notification function.

```
enum {
    kFolderManagerNotificationMessageUserLogIn = 'log+',
    kFolderManagerNotificationMessagePreUserLogIn = 'logj',
    kFolderManagerNotificationMessageUserLogOut = 'log-',
    kFolderManagerNotificationMessagePostUserLogOut = 'logp',
    kFolderManagerNotificationDiscardCachedData = 'dche',
    kFolderManagerNotificationMessageLoginStartup = 'stup'
};
```

Constants

`kFolderManagerNotificationMessageUserLogIn`
Sent when a user has logged in. When you receive this message `FindFolder` will return the `vRefNum` and `dirID` of the user's redirected folders until the user logs out. This message can be used to load the new user's preferences.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Folders.h`.

`kFolderManagerNotificationMessagePreUserLogIn`
Sent just prior to redirecting `FindFolder` to the user's folders. Calling `FindFolder` when receiving this notification will return the `vRefNum` and `dirID` of the system folders. This message can be used to update the owner's preference files prior to `FindFolder` being redirected.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Folders.h`.

`kFolderManagerNotificationMessageUserLogOut`
Sent when a user has logged out. This is the last time `FindFolder` will return the user's folders; after this notification `FindFolder` will return the `vRefNum` and `dirID` of system folders. This message can be used to update a user's preference files during logout.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Folders.h`.

`kFolderManagerNotificationMessagePostUserLogOut`
Sent just after `FindFolder` has been restored to return the `vRefNum` and `dirID` of system folders. This message can be used to load the owner's preferences.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Folders.h`.

`kFolderManagerNotificationDiscardCachedData`
Sent by third-party software when the entire Folder Manager cache should be flushed.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Folders.h`.

`kFolderManagerNotificationMessageLoginStartup`
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Folders.h`.

FolderManagerCallNotificationProcs Options

Used in the `options` parameter of `FolderManagerCallNotificationProcs`.

```
enum {
    kStopIfAnyNotificationProcReturnsErrorBit = 31
};
```

Result Codes

The most common result codes returned by Folder Manager are listed below.

Result Code	Value	Description
`badFolderDescErr`	-4270	Invalid folder Available in Mac OS X v10.0 and later.
`duplicateFolderDescErr`	-4271	Duplicate folders for a particular routing Available in Mac OS X v10.0 and later.
`noMoreFolderDescErr`	-4272	Available in Mac OS X v10.0 and later.
`invalidFolderTypeErr`	-4273	Invalid folder name Available in Mac OS X v10.0 and later.
`duplicateRoutingErr`	-4274	Same routing for two folders Available in Mac OS X v10.0 and later.
`routingNotFoundErr`	-4275	No routing set up for the folder passed in Available in Mac OS X v10.0 and later.
`badRoutingSizeErr`	-4276	Incorrect `descSize` field of the folder routing structure Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availability information by using the Folder Manager selectors defined in the Gestalt Manager. For more information see *Inside Mac OS X: Gestalt Manager Reference*.

Gestalt Manager Reference

Framework: CoreServices/CoreServices.h

Declared in Gestalt.h

Overview

You can use the Gestalt Manager and other system software facilities to investigate the operating environment. You need to know about the operating environment if your application takes advantage of hardware or software that is not available on all Macintosh computers. You can also use the Gestalt Manager to inform the operating system that your software is present and to find out about other software registered with the Gestalt Manager.

Carbon supports the Gestalt Manager. However, the results returned by Gestalt functions in Mac OS X are relevant only to your application's context. In general, the `Gestalt` function returns a different result when called from a Carbon application running in Mac OS X than it returns when called from a Classic application in Mac OS X, because these are different environments. For example, Carbon does not use a ROM, so calling `Gestalt` from a Carbon application on a beige G3 Macintosh computer and passing the `ROMVersion` selector returns a different result than `Gestalt` returns for a Classic application on the same computer. In fact, `Gestalt` could conceivably return different results for the same call by two Carbon applications.

Because `Gestalt` operates on a per-context basis in Mac OS X, you can't use it to share information (through pointers or any other means) among applications.

The `ROMVersion` and `machineType` selectors are not supported in Carbon.

In versions of the Mac OS prior to Mac OS X, the `NewGestalt` and `ReplaceGestalt` functions make use of the system heap, so that new or replaced selectors are available to any process. In Mac OS X, however, there is no system heap, and the selectors are available only on a per-context basis.

Functions by Task

Getting and Setting Gestalt Selector Codes and Values

`Gestalt` (page 1005)
Obtains information about the operating environment.

`NewGestaltValue` (page 1008)
Installs a new `Gestalt` selector code and a value that `Gestalt` returns for that selector.

`SetGestaltValue` (page 1010)
Sets the value the function `Gestalt` will return for a specified selector code, installing the selector if it was not already installed.

`ReplaceGestaltValue` (page 1009)
Replaces the value that the function `Gestalt` returns for a specified selector code with the value provided to the function.

`DeleteGestaltValue` (page 1004)
Deletes a `Gestalt` selector code so that it is no longer recognized by `Gestalt`.

`NewGestalt` (page 1007) Deprecated in Mac OS X v10.3
Adds a selector code to those already recognized by `Gestalt`. (Deprecated. Use `NewGestaltValue` (page 1008) instead.)

`ReplaceGestalt` (page 1009) Deprecated in Mac OS X v10.3
Replaces the selector function associated with an existing selector code. (Deprecated. Use `NewGestaltValue` (page 1008) instead.)

Working With Universal Procedure Pointers for Gestalt Selector Functions

`NewSelectorFunctionUPP` (page 1008)
Creates a universal procedure pointer (UPP) to a selector callback function.

`DisposeSelectorFunctionUPP` (page 1005)
Disposes of a universal procedure pointer to a selector callback function.

`InvokeSelectorFunctionUPP` (page 1006)
Invokes a selector callback function.

Functions

DeleteGestaltValue

Deletes a `Gestalt` selector code so that it is no longer recognized by `Gestalt`.

```
OSErr DeleteGestaltValue (
   OSType selector
);
```

Parameters

selector
The selector code you want to delete. This should be a four-character sequence similar to those defined in "Gestalt Manager Constants" (page 1012).

Return Value
A result code. See "Gestalt Manager Result Codes" (page 1113).

Discussion
After calling this function, subsequent query or replacement calls for the selector code will fail as if the selector had never been installed.

In Mac OS X, the selector is on a per-context basis. You cannot use this function to affect another process.

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

Declared In
`Gestalt.h`

DisposeSelectorFunctionUPP

Disposes of a universal procedure pointer to a selector callback function.

```
void DisposeSelectorFunctionUPP (
   SelectorFunctionUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer you want to dispose of.

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

Declared In
`Gestalt.h`

Gestalt

Obtains information about the operating environment.

```
OSErr Gestalt (
   OSType selector,
   SInt32 *response
);
```

Parameters

selector

The selector code for the information you need. You can provide any of the four-character sequences defined in "Gestalt Manager Constants" (page 1012).

response

On input, `Gestalt` interprets this parameter as an address at which it is to place the result returned by the selector function. `Gestalt` ignores any information already at this address.

On return, a pointer to the requested information whose format depends on the selector code specified in the selector parameter. Note that the `Gestalt` function returns the response from all selectors in a long word, which occupies 4 bytes. When not all 4 bytes are needed, the significant information appears in the low-order byte or bytes.

Return Value
A result code. See "Gestalt Manager Result Codes" (page 1113).

Discussion
The Apple-defined selector codes fall into two categories: environmental selectors, which supply specific environmental information you can use to control the behavior of your application, and informational selectors, which can't supply information you can use to determine what hardware or software features are available. You can use one of the selector codes defined by Apple or a selector code defined by a third-party product.

Selectors with the suffix `Attr` return a 32-bit response value in which the individual bits represent specific attributes. The constants listed for these response values represent bit numbers.

Special Considerations

When passed one of the Apple-defined selector codes, the `Gestalt` function does not move or purge memory and therefore may be called even at interrupt time. However, selector functions associated with non-Apple selector codes might move or purge memory, and third-party software can alter the Apple-defined selector functions. Therefore, it is safest always to assume that `Gestalt` could move or purge memory.

Version Notes

The `ROMVersion` and `machineType` selectors are not supported in Carbon.

In general, the `Gestalt` function returns a different result when called from a Carbon application running in Mac OS X than it returns when called from a Classic application in Mac OS X, because these are different environments. For example, Carbon does not use a ROM, so calling `Gestalt` from a Carbon application on a beige G3 Macintosh computer and passing the `ROMVersion` selector returns a different result than Gestalt returns for a Classic application on the same computer.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTCarbonShell
QTMetaData
SampleScannerApp
SoftVDigX
WhackedTV

Declared In

`Gestalt.h`

InvokeSelectorFunctionUPP

Invokes a selector callback function.

```
OSErr InvokeSelectorFunctionUPP (
   OSType selector,
   long *response,
   SelectorFunctionUPP userUPP
);
```

Parameters

selector

The selector code for the function you want to invoke. You can provide any of the four-character sequences defined in "Gestalt Manager Constants" (page 1012).

response

On ouput, the value associated with the selector code.

userUPP

A universal procedure pointer to the selector callback function you want to invoke.

Return Value

A result code. See "Gestalt Manager Result Codes" (page 1113).

Discussion
You should not need to call this function, as the operating system invokes your selecor callback for you.

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

Declared In
`Gestalt.h`

NewGestalt

Adds a selector code to those already recognized by `Gestalt`. (Deprecated in Mac OS X v10.3. Use `NewGestaltValue` (page 1008) instead.)

```
OSErr NewGestalt (
   OSType selector,
   SelectorFunctionUPP gestaltFunction
);
```

Parameters

selector

The selector code you want to add. This should be a four-character sequence similar to those defined in "Gestalt Manager Constants" (page 1012).

gestaltFunction

A universal procedure pointer (UPP) to the selector callback function that `Gestalt` executes when it receives the new selector code. See `SelectorFunctionProcPtr` (page 1011) for more information on the callback you need to provide.

Return Value
A result code. See "Gestalt Manager Result Codes" (page 1113).

Discussion
The `NewGestalt` function registers a specified selector code with the Gestalt Manager so that when the `Gestalt` function is called with that selector code, the specified selector function is executed. Before calling `NewGestalt`, you must define a selector function callback. See `SelectorFunctionProcPtr` (page 1011) for a description of how to define your selector function.

Registering with the Gestalt Manager is a way for software such as system extensions to make their presence known to potential users of their services.

Special Considerations

You should avoid using the `NewGestalt` function to add a selector code, which requires moving your selector function into the system heap. Applications do not have access to the system heap 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
`Gestalt.h`

NewGestaltValue

Installs a new `Gestalt` selector code and a value that `Gestalt` returns for that selector.

```
OSErr NewGestaltValue (
   OSType selector,
   SInt32 newValue
);
```

Parameters

selector

The selector code you want to add. This should be a four-character sequence similar to those defined in "Gestalt Manager Constants" (page 1012).

newValue

The value to return for the new selector code.

Return Value

A result code. See "Gestalt Manager Result Codes" (page 1113).

Discussion

You call the function `NewGestaltValue` when the specified selector is not already installed and you don't want to override an existing value.

In Mac OS X, the new selector and value are on a per-context basis. That means they are available only to the application or other code that installs them. You cannot use this function to make information available to another process.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Gestalt.h`

NewSelectorFunctionUPP

Creates a universal procedure pointer (UPP) to a selector callback function.

```
SelectorFunctionUPP NewSelectorFunctionUPP (
   SelectorFunctionProcPtr userRoutine
);
```

Parameters

userRoutine

The address of the selector callback function.

Return Value

On return, a universal procedure pointer to the selector callback function. See the description of the `SelectorFunctionUPP` data type.

Discussion

You use the `NewSelectorFunctionUPP` function to create a UPP to pass to the `NewGestalt` or `ReplaceGestalt` functions.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Gestalt.h`

ReplaceGestalt

Replaces the selector function associated with an existing selector code. (Deprecated in Mac OS X v10.3. Use `NewGestaltValue` (page 1008) instead.)

```
OSErr ReplaceGestalt (
   OSType selector,
   SelectorFunctionUPP gestaltFunction,
   SelectorFunctionUPP *oldGestaltFunction
);
```

Parameters

selector

The selector code for the function you want to replace. You must provide the four-character sequence you provided previously for the function you are replacing.

gestaltFunction

A universal procedure pointer to the replacement selector function. You must obtain the value for this argument by calling the `NewGestaltSelectorFunctionUPP` function.

oldGestaltFunction

On output, a universal procedure pointer to the callback function previously associated with the specified selector. If the function `ReplaceGestalt` returns an error of any type, then the value of `oldGestaltFunction` is undefined.

Return Value
A result code. See "Gestalt Manager Result Codes" (page 1113).

Special Considerations

You should avoid using the `ReplaceGestalt` function to replace an existing selector callback function, which also requires your replacement function to reside in the system heap. Applications do not have access to the system heap 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
`Gestalt.h`

ReplaceGestaltValue

Replaces the value that the function `Gestalt` returns for a specified selector code with the value provided to the function.

```
OSErr ReplaceGestaltValue (
   OSType selector,
   SInt32 replacementValue
);
```

Parameters

selector

The selector code you want to add. This should be a four-character sequence similar to those defined in "Gestalt Manager Constants" (page 1012).

replacementValue

The replacement `Gestalt` value for the selector code.

Return Value

A result code. See "Gestalt Manager Result Codes" (page 1113).

Discussion

You use the function `ReplaceGestaltValue` to replace an existing value. You should not call this function to introduce a value that doesn't already exist; instead call the function `NewGestaltValue` (page 1008).

In Mac OS X, the selector and replacement value are on a per-context basis. That means they are available only to the application or other code that installs them. You cannot use this function to make information available to another process.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Gestalt.h`

SetGestaltValue

Sets the value the function `Gestalt` will return for a specified selector code, installing the selector if it was not already installed.

```
OSErr SetGestaltValue (
   OSType selector,
   SInt32 newValue
);
```

Parameters

selector

The selector code you want to set. This should be a four-character sequence similar to those defined in "Gestalt Manager Constants" (page 1012).

newValue

The new `Gestalt` value for the selector code.

Return Value

A result code. See "Gestalt Manager Result Codes" (page 1113).

Discussion

You use `SetGestaltValue` to establish a value for a selector, without regard to whether the selector was already installed.

In Mac OS X, the selector and new value are on a per-context basis. That means they are available only to the application or other code that installs them. You cannot use this function to make information available to another process.

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

Declared In
`Gestalt.h`

Callbacks

SelectorFunctionProcPtr

Defines a pointer to a selector callback function that returns information associated with your own selector code.

```
typedef OSErr (*SelectorFunctionProcPtr)
(
    OSType selector,
    long * response
);
```

If you name your function `MySelectorFunctionProc`, you would declare it like this:

```
OSErr SelectorFunctionProcPtr (
    OSType selector,
    long * response
);
```

Parameters

`selector`
The selector code that triggers the function. This should be a four-character sequence similar to those defined in "Gestalt Manager Constants" (page 1012).

`response`
On output, the information associated with the selector code.

Return Value
A result code. See "Gestalt Manager Result Codes" (page 1113).

Discussion
Your selector function places the requested information in the `response` parameter and returns a result code. If the information is not available, the selector function returns the appropriate error code, which the `Gestalt` function returns as its function result.

A selector function can call `Gestalt` or even other selector functions.

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

Declared In
`Gestalt.h`

Data Types

SelectorFunctionUPP

Defines a universal procedure pointer to a selector function callback.

```
typedef SelectorFunctionProcPtr SelectorFunctionUPP;
```

Discussion
You can obtain a `SelectorFunctionUPP` by calling the function `NewSelectorFunctionUPP` (page 1008). For more information, see `SelectorFunctionProcPtr` (page 1011).

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

Declared In
`Gestalt.h`

Constants

Addressing Mode Attribute Selectors

Specify feature availability information for the addressing mode of the operating system.

```
enum {
    gestaltAddressingModeAttr = 'addr',
    gestalt32BitAddressing = 0,
    gestalt32BitSysZone = 1,
    gestalt32BitCapable = 2
};
```

Constants

`gestaltAddressingModeAttr`
The `Gestalt` selector you pass to determine the addressing mode attributes that are present.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestalt32BitAddressing`
If `true`, the operating system is using 32-bit addressing mode.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestalt32BitSysZone`
If `true`, there is a 32-bit compatible system zone.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestalt32BitCapable`
If `true`, Machine is 32-bit capable.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion
Before calling any function dependent on memory, your application should pass the selector `gestaltAddressingModeAttr` to the Gestalt function to determine the addressing mode attributes that are present.

Admin Attribute Selectors

Specify feature availability for Macintosh Manager adminstration software.

```
enum {
    gestaltAdminFeaturesFlagsAttr = 'fred',
    gestaltFinderUsesSpecialOpenFoldersFile = 0
};
```

Constants
`gestaltAdminFeaturesFlagsAttr`
The `Gestalt` selector you pass to determine the admin features that are present. This selector is typically used by the system.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltFinderUsesSpecialOpenFoldersFile`
Specifies that the Finder uses a special file to store the list of open folders.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

AFP Client Selectors

Specify version and feature availability information for the AFP client.

```
enum {
    gestaltAFPClient = 'afps',
    gestaltAFPClientVersionMask = 0x0000FFFF,
    gestaltAFPClient3_5 = 0x0001,
    gestaltAFPClient3_6 = 0x0002,
    gestaltAFPClient3_6_1 = 0x0003,
    gestaltAFPClient3_6_2 = 0x0004,
    gestaltAFPClient3_6_3 = 0x0005,
    gestaltAFPClient3_7 = 0x0006,
    gestaltAFPClient3_7_2 = 0x0007,
    gestaltAFPClient3_8 = 0x0008,
    gestaltAFPClient3_8_1 = 0x0009,
    gestaltAFPClient3_8_3 = 0x000A,
    gestaltAFPClient3_8_4 = 0x000B,
    gestaltAFPClientAttributeMask = 0xFFFF0000,
    gestaltAFPClientCfgRsrc = 16,
    gestaltAFPClientSupportsIP = 29,
    gestaltAFPClientVMUI = 30,
    gestaltAFPClientMultiReq = 31
};
```

Alias Manager Attribute Selectors

Specify feature availability information for the Alias Manager.

```
enum {
    gestaltAliasMgrAttr = 'alis',
    gestaltAliasMgrPresent = 0,
    gestaltAliasMgrSupportsRemoteAppletalk = 1,
    gestaltAliasMgrSupportsAOCEKeychain = 2,
    gestaltAliasMgrResolveAliasFileWithMountOptions = 3,
    gestaltAliasMgrFollowsAliasesWhenResolving = 4,
    gestaltAliasMgrSupportsExtendedCalls = 5,
    gestaltAliasMgrSupportsFSCalls = 6,
    gestaltAliasMgrPrefersPath = 7
};
```

Constants

`gestaltAliasMgrAttr`

The selector you pass to the `Gestalt` function to determine the Alias Manager attributes.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Appearance Manager Attribute Selectors

Specify feature availability information for the Appearance Manager.

```
enum {
    gestaltAppearanceAttr = 'appr',
    gestaltAppearanceExists = 0,
    gestaltAppearanceCompatMode = 1
};
```

Constants

`gestaltAppearanceAttr`

The `Gestalt` selector passed to determine whether the Appearance Manager is present. Produces a 32-bit value whose bits you should test to determine which Appearance Manager features are available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltAppearanceExists`

If this bit is set, Appearance Manager functions are available. To determine which version of the Appearance Manager is installed, check for the presence of the `Gestalt` selector `gestaltAppearanceVersion`. If this bit is not set, Appearance Manager functions are not available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltAppearanceCompatMode`

If this bit is set, systemwide platinum appearance is off. When systemwide platinum appearance is off, the Appearance Manager does not auto-map standard System 7 definition functions to their Mac OS 8 equivalents (for those applications that have not called `RegisterAppearanceClient`). If this bit is not set, systemwide platinum appearance is on, and the Appearance Manager auto-maps standard System 7 definition functions to their Mac OS 8 equivalents for all applications.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion

Before calling any functions dependent upon the Appearance Manager's presence, your application should pass the selector `gestaltAppearanceAttr` to the `Gestalt` function to determine whether the Appearance Manager is present. To determine which version of the Appearance Manager is installed, your application should check for the presence of the `Gestalt` selector `gestaltAppearanceVersion`.

Appearance Manager Version Selector

Specifies version information for the Appearance Manager.

```
enum {
    gestaltAppearanceVersion = 'apvr'
};
```

Constants

`gestaltAppearanceVersion`

The `Gestalt` selector passed to determine which version of the Appearance Manager is installed. If this selector exists, Appearance Manager 1.0.1 (or later) is installed. The version number of the currently installed Appearance Manager is returned in the low-order word of the result in binary code decimal format (for example, version 1.0.1 would be 0x0101). If this selector does not exist but `gestaltAppearanceAttr` does, Appearance Manager 1.0 is installed.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Apple Event Manager Attribute Selectors

Specify feature availability information for the Apple Event Manager.

```
enum {
    gestaltAppleEventsAttr = 'evnt',
    gestaltAppleEventsPresent = 0,
    gestaltScriptingSupport = 1,
    gestaltOSLInSystem = 2,
    gestaltSupportsApplicationURL = 4
};
```

Constants

`gestaltAppleEventsAttr`
A selector you pass to the `Gestalt` function. If the Apple Event Manager is not present, the `Gestalt` function returns an error value; otherwise, it returns `noErr` and supplies, in the `response` parameter, a 32-bit value whose bits specify which features of the Apple Event Manager are available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltAppleEventsPresent`
A `Gestalt` attribute constant. If the bit specified by this constant is set in the `response` parameter value supplied by `Gestalt` for the `gestaltAppleEventsAttr` selector, the Apple Event Manager is present and installed in the system.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltScriptingSupport`
A `Gestalt` attribute constant. If the bit specified by this constant is set in the `response` parameter value supplied by `Gestalt` for the `gestaltAppleEventsAttr` selector, the Open Scripting Architecture (OSA) is available to provide scripting support. The OSA is described in "Scripting Components".

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltOSLInSystem`
A `Gestalt` attribute constant. If the bit specified by this constant is set in the `response` parameter value supplied by `Gestalt` for the `gestaltAppleEventsAttr` selector, the Object Support Library (OSL) is part of the system.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltSupportsApplicationURL`
Available in Mac OS X v10.1 and later.

Declared in `Gestalt.h`.

AppleScript Attribute Selectors

Specify feature availability information for AppleScript.

```
enum {
    gestaltAppleScriptAttr = 'ascr',
    gestaltAppleScriptPresent = 0,
    gestaltAppleScriptPowerPCSupport = 1
};
```

Constants

`gestaltAppleScriptAttr`

A selector you pass to the `Gestalt` function. If AppleScript is not present, the `Gestalt` function returns an error value; otherwise, it returns `noErr` and supplies, in the `response` parameter, a 32-bit value whose bits specify which AppleScript features are available.

The only bit currently in use specifies whether AppleScript is present. You can test this bit with the constant `gestaltAppleScriptPresent`.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltAppleScriptPresent`

A `Gestalt` attribute constant. If the bit specified by this constant is set in the `response` parameter value supplied by `Gestalt` for the `gestaltAppleScriptAttr` selector, AppleScript is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltAppleScriptPowerPCSupport`

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

AppleScript Version Selector

Specifies version information for AppleScript.

```
enum {
    gestaltAppleScriptVersion = 'ascv'
};
```

Constants

`gestaltAppleScriptVersion`

A selector you pass to the `Gestalt` function. If AppleScript is not present, the `Gestalt` function returns an error value; otherwise, it returns `noErr` and supplies, in the `response` parameter, a 32-bit AppleScript version number.

The low word of the 32-bit AppleScript version number specifies the current AppleScript version, while the high word specifies a compatibility version. For example, for AppleScript 1.3.7, which shipped with Mac OS 8.6, the value returned in the `response` parameter, viewed as a hex number, is 0x01100137. The low word, 0x0137, refers to the current AppleScript version. The high word, 0x0110, refers to the compatibility version number—scripts written for AppleScript versions 1.1.0 and later will run with AppleScript version 1.3.7.

The Version Notes section provides additional information about AppleScript versions and features.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Version Notes
For System 7.0 and 7.1, AppleScript and the Apple Event Manager were optional installs. Starting with System 7.5, they are part of a standard install, so if you've already checked for the presence of System 7.5 or later, you'll know that the Apple Event Manager is available, though AppleScript could be disabled using the Extensions Manager.

If you need features that are only available starting with a specific version of AppleScript, call `Gestalt` with the `gestaltAppleScriptVersion` selector to obtain the version number, then determine whether it is greater than or equal to the version your application requires.

AppleTalk Driver Version Selector

Specifies version information for the AppleTalk driver.

```
enum {
    gestaltATalkVersion = 'atkv'
};
```

Constants

`gestaltATalkVersion`

The version number of the AppleTalk driver, in the format introduced with AppleTalk version 56. The version is stored in the high 3 bytes of the return value.

Byte 3 contains the major revision number, byte 2 contains the minor revision number, and byte 1 contains a constant that represents the release stage.

For example, if you call the `Gestalt` function with the 'atkv' selector when AppleTalk version 57 is loaded, you receive the long integer response value $39008000.

Byte 0 always contains 0.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

AppleTalk Version Selector

Specifies version information for AppleTalk.

```
enum {
    gestaltAppleTalkVersion = 'atlk'
};
```

Constants

`gestaltAppleTalkVersion`

The version number of the AppleTalk driver (in particular, the .MPP driver) currently installed. The version number is placed into the low-order byte of the result; ignore the three high-order bytes. If an AppleTalk driver is not currently open, the `response` parameter is 0.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

ATSUI Attribute Selectors

Specify feature availability for Apple Type Services for Unicode Imaging.

```
enum {
    gestaltATSUFeatures = 'uisf',
    gestaltATSUTrackingFeature = 0x00000001,
    gestaltATSUMemoryFeature = 0x00000001,
    gestaltATSUFallbacksFeature = 0x00000001,
    gestaltATSUGlyphBoundsFeature = 0x00000001,
    gestaltATSULineControlFeature = 0x00000001,
    gestaltATSULayoutCreateAndCopyFeature = 0x00000001,
    gestaltATSULayoutCacheClearFeature = 0x00000001,
    gestaltATSUTextLocatorUsageFeature = 0x00000002,
    gestaltATSULowLevelOrigFeatures = 0x00000004,
    gestaltATSUFallbacksObjFeatures = 0x00000008,
    gestaltATSUIgnoreLeadingFeature = 0x00000008,
    gestaltATSUByCharacterClusterFeature = 0x00000010,
    gestaltATSUAscentDescentControlsFeature = 0x00000010,
    gestaltATSUHighlightInactiveTextFeature = 0x00000010,
    gestaltATSUPositionToCursorFeature = 0x00000010,
    gestaltATSUBatchBreakLinesFeature = 0x00000010,
    gestaltATSUTabSupportFeature = 0x00000010,
    gestaltATSUDirectAccess = 0x00000010,
    gestaltATSUDecimalTabFeature  = 0x00000020,
    gestaltATSUBiDiCursorPositionFeature = 0x00000020,
    gestaltATSUNearestCharLineBreakFeature = 0x00000020,
    gestaltATSUHighlightColorControlFeature = 0x00000020,
    gestaltATSUUnderlineOptionsStyleFeature = 0x00000020,
    gestaltATSUStrikeThroughStyleFeature = 0x00000020,
    gestaltATSUDropShadowStyleFeature = 0x00000020
};
```

Constants

`gestaltATSUFeatures`

Specifies the ATSUI features available on the user's system. You pass this selector to the `Gestalt` function. On return, the `Gestalt` function passes back a value that represents the features available in the version of ATSUI installed on the user's system.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltATSUTrackingFeature`

If the bit specified by this mask constant is set, the functions `ATSUCountFontTracking` and `ATSUGetIndFontTracking` are available.

Available beginning with ATSUI 1.1.

Declared in `Gestalt.h`.

`gestaltATSUMemoryFeature`

If the bit specified by this mask is set, the functions `ATSUCreateMemorySetting`, `ATSUSetCurrentMemorySetting`, `ATSUGetCurrentMemorySetting`, and `ATSUDisposeMemorySetting` are available.

Available beginning with ATSUI 1.1.

Declared in `Gestalt.h`.

`gestaltATSUFallbacksFeature`

If the bit specified by this mask is set, the functions `ATSUSetFontFallbacks` and `ATSUGetFontFallbacks` are available.

Available beginning with ATSUI 1.1.

Declared in `Gestalt.h`.

`gestaltATSUGlyphBoundsFeature`
If the bit specified by this mask is set, the function `ATSUGetGlyphBounds` is available.

Available beginning with ATSU 1.1.

Declared in `Gestalt.h`.

`gestaltATSULineControlFeature`
If the bit specified by this mask is set, the functions `ATSUCopyLineControls`, `ATSUSetLineControls`, `ATSUGetLineControl`, `ATSUGetAllLineControls`, and `ATSUClearLineControls` are available.

Available beginning with ATSUI 1.1.

Declared in `Gestalt.h`.

`gestaltATSULayoutCreateAndCopyFeature`
If the bit specified by this mask is set, the function `ATSUCreateAndCopyTextLayout` is available.

Available beginning with ATSUI 1.1.

Declared in `Gestalt.h`.

`gestaltATSULayoutCacheClearFeature`
If the bit specified by this mask is set, the function `ATSUClearLayoutCache` is available.

Available beginning with ATSUI 1.1.

Declared in `Gestalt.h`.

`gestaltATSUTextLocatorUsageFeature`
If the bit specified by this mask is set, the text-break locator attribute is available for both style and text layout objects.

Available beginning with ATSUI 1.2.

Declared in `Gestalt.h`.

`gestaltATSULowLevelOrigFeatures`
If the bit specified by this mask is set, the low-level features introduced in ATSUI version 2.0 are available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltATSUFallbacksObjFeatures`
If the bit specified by this mask is set, `ATSUFontFallbacks` objects are available.

Available beginning with ATSUI version 2.3.

Declared in `Gestalt.h`.

`gestaltATSUIgnoreLeadingFeature`
If the bit specified by this mask is set, the line layout option (`kATSIgnoreFontLeadingTag`) to ignore the font leading value is available.

Available beginning with ATSUI version 2.3.

Declared in `Gestalt.h`.

`gestaltATSUByCharacterClusterFeature`
If the bit specified by this mask is set, ATSUI cursor movement types are available.

Available beginning with ATSUI version 2.4.

Declared in `Gestalt.h`.

`gestaltATSUAscentDescentControlsFeature`
If the bit specified by this mask is set, ascent and descent controls (`kATSUDescentTag` and `kATSUAscentTag`) are available.

Available beginning with ATSUI version 2.4.

Declared in `Gestalt.h`.

`gestaltATSUHighlightInactiveTextFeature`
If the bit specified by this mask is set, the highlight inactive text feature is available.

Available beginning with ATSUI version 2.4.

Declared in `Gestalt.h`.

`gestaltATSUPositionToCursorFeature`
If the bit specified by this mask is set, the position-to-cursor feature is available.

Available beginning with ATSUI version 2.4.

Declared in `Gestalt.h`.

`gestaltATSUBatchBreakLinesFeature`
If the bit specified by this mask is set, the `ATSUBatchBreakLines` function is available.

Available beginning with ATSUI version 2.4.

Declared in `Gestalt.h`.

`gestaltATSUTabSupportFeature`
If the bit specified by this mask is set, support for tabs is available.

Available beginning with ATSUI version 2.4.

Declared in `Gestalt.h`.

`gestaltATSUDirectAccess`
If the bit specified by this mask is set, ATSU direct-access functions are available. These functions let you access glyph information directly.

Available beginning with ATSUI version 2.4.

Declared in `Gestalt.h`.

`gestaltATSUDecimalTabFeature`
If the bit specified by this mask is set, your application can set a decimal tab character.

Available beginning with ATSUI version 2.5.

Declared in `Gestalt.h`.

`gestaltATSUBiDiCursorPositionFeature`
If the bit specified by this mask is set, support for bidirectional cursor positioning is available.

Available beginning with ATSUI version 2.5.

Declared in `Gestalt.h`.

`gestaltATSUNearestCharLineBreakFeature`
If the bit specified by this mask is set, the nearest character line break feature is available.

Available beginning with ATSUI version 2.5.

Declared in `Gestalt.h`.

`gestaltATSUHighlightColorControlFeature`
If the bit specified by this mask is set, your application can control highlight color.

Available beginning with ATSUI version 2.5.

Declared in `Gestalt.h`.

`gestaltATSUUnderlineOptionsStyleFeature`
If the bit specified by this mask is set, underline options are available.

Available beginning with ATSUI version 2.5.

Declared in `Gestalt.h`.

`gestaltATSUStrikeThroughStyleFeature`
If the bit specified by this mask is set, strike through styles are available.

Available beginning with ATSUI version 2.5.

Declared in `Gestalt.h`.

`gestaltATSUDropShadowStyleFeature`
If the bit specified by this mask is set, drop shadow features are available.

Available beginning with ATSUI version 2.5.

Declared in `Gestalt.h`.

Discussion
You can pass the `gestaltATSUFeature` selector to the `Gestalt` function to obtain a value that specifies which ATSUI features are available on the user's system.

You can pass the `gestaltATSUVersion` selector to the `Gestalt` function to determine which version of ATSUI is installed on the user's system. See "ATSUI Version Selectors" (page 1022) for more information

ATSUI Version Selectors

Specify version information for Apple Type Service for Unicode Imaging.

```
enum {
    gestaltATSUVersion = 'uisv',
    gestaltOriginalATSUVersion = (1 << 16),
    gestaltATSUUpdate1 = (2 << 16),
    gestaltATSUUpdate2 = (3 << 16),
    gestaltATSUUpdate3 = (4 << 16),
    gestaltATSUUpdate4 = (5 << 16),
    gestaltATSUUpdate5 = (6 << 16),
    gestaltATSUUpdate6 = (7 << 16),
    gestaltATSUUpdate7 = (8 << 16)
};
```

Constants
`gestaltATSUVersion`
Specifies the version of ATSUI installed on the user's system. You pass this selector to the `Gestalt` function. On return, the `Gestalt` function passes back a value that represents the version of ATSUI installed on the user's system.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltOriginalATSUVersion`
Indicates that version 1.0 of ATSUI is installed on the user's system.

Available beginning with ATSUI 1.0.

Declared in `Gestalt.h`.

`gestaltATSUUpdate1`
Indicates that version 1.1 of ATSUI is installed on the user's system.

Available beginning with ATSUI 1.1.

Declared in `Gestalt.h`.

`gestaltATSUUpdate2`
Indicates that version 1.2 of ATSUI is installed on the user's system.

Available beginning with ATSUI 1.2.

Declared in `Gestalt.h`.

`gestaltATSUUpdate3`
Indicates that version 2.0 of ATSUI is installed on the user's system.

Available beginning with ATSUI 2.0.

Declared in `Gestalt.h`.

`gestaltATSUUpdate4`
Indicates that ATSUI for a version of Mac OS X from 10.0.1 through 10.0.4 is installed on the user's system.

Available beginning with Mac OS X version 10.0.1.

Declared in `Gestalt.h`.

`gestaltATSUUpdate5`
Indicates that version 2.3 of ATSUI is installed on the user's system. Available beginning with ATSUI 2.3, in Mac OS X version 10.1.

Available in Mac OS X v10.1 and later.

Declared in `Gestalt.h`.

`gestaltATSUUpdate6`
Indicates that version 2.4 of ATSUI is installed on the user's system. Available beginning with ATSUI 2.4, in Mac OS X version 10.2.

Available in Mac OS X v10.2 and later.

Declared in `Gestalt.h`.

`gestaltATSUUpdate7`
Indicates that version 2.5 of ATSUI is installed on the user's system. Available beginning with ATSUI 2.5, in Mac OS X version 10.3.

Available in Mac OS X v10.3 and later.

Declared in `Gestalt.h`.

Discussion
Before calling any functions dependent upon ATSUI, you should pass the `gestaltATSUVersion` selector to the `Gestalt` function to determine which version of ATSUI is available.

You can pass the `gestaltATSUFeatures` selector to the `Gestalt` function to determine which features of ATSUI are available. See "ATSUI Attribute Selectors" (page 1018) for more information.

ATA Manager Attribute Selectors

Specify feature availability information for the ATA Manager.

```
enum {
    gestaltATAAttr = 'ata ',
    gestaltATAPresent = 0
};
```

AUX Version Selector

Specifies version information for A/UX.

```
enum {
    gestaltAUXVersion = 'a/ux'
};
```

Constants

`gestaltAUXVersion`

The version of A/UX if it is currently executing. The result is placed into the low-order word of the response parameter. If A/UX is not executing, the `Gestalt` function returns `gestaltUnknownErr`.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

AVL Tree Attribute Selectors

Specify feature availability information for AVL tree routines.

```
enum {
    gestaltAVLTreeAttr = 'tree',
    gestaltAVLTreePresentBit = 0,
    gestaltAVLTreeSupportsHandleBasedTreeBit = 1,
    gestaltAVLTreeSupportsTreeLockingBit = 2
};
```

Bus Clock Version Selector

Specifies version information for the bus clock speed.

```
enum {
    gestaltBusClkSpeed = 'bclk'
};
```

Carbon Version Selector

Specifies version information for Carbon.

```
enum {
    gestaltCarbonVersion = 'cbon'
};
```

Classic Compatibility Attribute Selectors

Specify feature availability for the Classic environment.

```
enum {
    gestaltMacOSCompatibilityBoxAttr = 'bbox',
    gestaltMacOSCompatibilityBoxPresent = 0,
    gestaltMacOSCompatibilityBoxHasSerial = 1,
    gestaltMacOSCompatibilityBoxless = 2
};
```

CloseView Attribute Selectors

Specify feature availability information for CloseView.

```
enum {
    gestaltCloseViewAttr = 'BSDa',
    gestaltCloseViewEnabled = 0,
    gestaltCloseViewDisplayMgrFriendly = 1
};
```

Code Fragment Manager Attribute Selectors

Specify feature availability information for the Code Fragment Manager.

```
enum {
    gestaltCFMAttr = 'cfrg',
    gestaltCFMPresent = 0,
    gestaltCFMPresentMask = 0x0001,
    gestaltCFM99Present = 2,
    gestaltCFM99PresentMask = 0x0004
};
```

Collection Manager Version Selector

Specify version information for the Collection manager.

```
enum {
    gestaltCollectionMgrVersion = 'cltn'
};
```

Constants

`gestaltCollectionMgrVersion`
Collection Manager version.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Color Picker Version Selectors

Specify version information for the Color Picker.

```
enum {
    gestaltColorPickerVersion = 'cpkr',
    gestaltColorPicker = 'cpkr'
};
```

Discussion
To test for the availability and version of the Color Picker Manager, use the `Gestalt` function with the selector defined by this enumerator.

If the `Gestalt` function returns a value of 00000200, version 2.0 of the Color Picker Manager is available. If the `Gestalt` function returns a value of 00000100, version 1.0 (that is, the original Color Picker Package) is available.

ColorSync Manager Attribute Selectors

Specify feature availability information for the ColorSync Manager.

```
enum {
    gestaltColorMatchingAttr = 'cmta',
    gestaltHighLevelMatching = 0,
    gestaltColorMatchingLibLoaded = 1
};
```

Constants

`gestaltColorMatchingAttr`
The selector for obtaining version information. Use when calling the `Gestalt` function to check for particular ColorSync Manager features.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHighLevelMatching`
This constant is provided for backward compatibility only. Bit 0 of the `Gestalt` response value is always set if ColorSync is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltColorMatchingLibLoaded`
This constant is provided for backward compatibility only. Bit 1 of the `Gestalt` response value is always set on a Power Macintosh machine if ColorSync is present. It is always cleared on a 68K machine if ColorSync is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion
These constants were added to ColorSync version 2.0 to aid in the transition from 68K to PowerPC systems. They are not recommended for new applications and are not guaranteed to be carried forward in future versions of ColorSync. However, they are still supported as of version 2.5 for backward compatibility. If you call the `Gestalt` function passing the selector `gestaltColorMatchingAttr`, you can test the bit fields of

the returned value with the `gestaltColorMatchingLibLoaded` constant to determine if the ColorSync Manager shared libraries are loaded, or with the `gestaltHighLevelMatching` constant to determine if the ColorSync QuickDraw-specific functions are present.

ColorSync Manager Version Selectors

Specify version information for the ColorSync Manager.

```
enum {
    gestaltColorMatchingVersion = 'cmtc',
    gestaltColorSync10 = 0x0100,
    gestaltColorSync11 = 0x0110,
    gestaltColorSync104 = 0x0104,
    gestaltColorSync105 = 0x0105,
    gestaltColorSync20 = 0x0200,
    gestaltColorSync21 = 0x0210,
    gestaltColorSync211 = 0x0211,
    gestaltColorSync212 = 0x0212,
    gestaltColorSync213 = 0x0213,
    gestaltColorSync25 = 0x0250,
    gestaltColorSync26 = 0x0260,
    gestaltColorSync261 = 0x0261,
    gestaltColorSync30 = 0x0300
};
```

Constants

`gestaltColorMatchingVersion`
The selector for obtaining version information. Use when calling the `Gestalt` function to determine whether the ColorSync Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltColorSync10`
A `Gestalt` response value of `gestaltColorSync10` indicates version 1.0 of the ColorSync Manager is present. This version supports general purpose color matching only and does not provide QuickDraw-specific matching functions.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltColorSync11`
A `Gestalt` response value of `gestaltColorSync11` indicates version 1.0.3 of the ColorSync Manager is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltColorSync104`
A `Gestalt` response value of `gestaltColorSync104` indicates version 1.4 of the ColorSync Manager is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltColorSync105`
A `Gestalt` response value of `gestaltColorSync105` indicates version 1.5 of the ColorSync Manager is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltColorSync20`
A `Gestalt` response value of `gestaltColorSync20` indicates version 2.0 of the ColorSync Manager is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltColorSync21`
A `Gestalt` response value of `gestaltColorSync21` indicates version 2.1 of the ColorSync Manager is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltColorSync211`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltColorSync212`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltColorSync213`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltColorSync25`
A `Gestalt` response value of `gestaltColorSync25` indicates version 2.5 of the ColorSync Manager is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltColorSync26`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltColorSync261`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltColorSync30`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion
These constants were added to ColorSync version 2.0 to aid in the transition from 68K to PowerPC systems. They are not recommended for new applications and are not guaranteed to be carried forward in future versions of ColorSync. However, they are still supported as of version 2.5 for backward compatibility. If you call the `Gestalt` function passing the selector `gestaltColorMatchingAttr`, you can test the bit fields of

the returned value with the `gestaltColorMatchingLibLoaded` constant to determine if the ColorSync Manager shared libraries are loaded, or with the `gestaltHighLevelMatching` constant to determine if the ColorSync QuickDraw-specific functions are present.

Communications Toolbox Version Selector

Specifies version information for the Communications Toolbox.

```
enum {
    gestaltCTBVersion = 'ctbv'
};
```

Constants

`gestaltCTBVersion`

The version number of the Communications Toolbox (in the low-order word of the return value).

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Communication Resource Manager Attribute Selectors

Specify version and feature availability information for the Communications Resource Manager.

```
enum {
    gestaltCRMAttr = 'crm ',
    gestaltCRMPresent = 0,
    gestaltCRMPersistentFix = 1,
    gestaltCRMToolRsrcCalls = 2
};
```

Component Manager Version Selectors

Specify version information for the Coomponent Manager.

```
enum {
    gestaltComponentMgr = 'cpnt',
    gestaltComponentPlatform = 'copl'
};
```

Constants

`gestaltComponentMgr`

The `Gestalt` selector you pass to determine what version of the Component Manager is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion

To determine the version of the current Component Manager, your application should pass the selector `gestaltComponentMgr` to the `Gestalt` function.

Computer Model Selectors

Specify computer models.

```
enum {
    gestaltMachineType = 'mach',
    gestaltClassic = 1,
    gestaltMacXL = 2,
    gestaltMac512KE = 3,
    gestaltMacPlus = 4,
    gestaltMacSE = 5,
    gestaltMacII = 6,
    gestaltMacIIx = 7,
    gestaltMacIIcx = 8,
    gestaltMacSE030 = 9,
    gestaltPortable = 10,
    gestaltMacIIci = 11,
    gestaltPowerMac8100_120 = 12,
    gestaltMacIIfx = 13,
    gestaltMacClassic = 17,
    gestaltMacIIsi = 18,
    gestaltMacLC = 19,
    gestaltMacQuadra900 = 20,
    gestaltPowerBook170 = 21,
    gestaltMacQuadra700 = 22,
    gestaltClassicII = 23,
    gestaltPowerBook100 = 24,
    gestaltPowerBook140 = 25,
    gestaltMacQuadra950 = 26,
    gestaltMacLCIII = 27,
    gestaltPerforma450 = gestaltMacLCIII,
    gestaltPowerBookDuo210 = 29,
    gestaltMacCentris650 = 30,
    gestaltPowerBookDuo230 = 32,
    gestaltPowerBook180 = 33,
    gestaltPowerBook160 = 34,
    gestaltMacQuadra800 = 35,
    gestaltMacQuadra650 = 36,
    gestaltMacLCII = 37,
    gestaltPowerBookDuo250 = 38,
    gestaltAWS9150_80 = 39,
    gestaltPowerMac8100_110 = 40,
    gestaltAWS8150_110 = gestaltPowerMac8100_110,
    gestaltPowerMac5200 = 41,
    gestaltPowerMac5260 = gestaltPowerMac5200,
    gestaltPerforma5300 = gestaltPowerMac5200,
    gestaltPowerMac6200 = 42,
    gestaltPerforma6300 = gestaltPowerMac6200,
    gestaltMacIIvi = 44,
    gestaltMacIIvm = 45,
    gestaltPerforma600 = gestaltMacIIvm,
    gestaltPowerMac7100_80 = 47,
    gestaltMacIIvx = 48,
    gestaltMacColorClassic = 49,
    gestaltPerforma250 = gestaltMacColorClassic,
    gestaltPowerBook165c = 50,
    gestaltMacCentris610 = 52,
    gestaltMacQuadra610 = 53,
    gestaltPowerBook145 = 54,
    gestaltPowerMac8100_100 = 55,
    gestaltMacLC520 = 56,
    gestaltAWS9150_120 = 57,
```

```
    gestaltPowerMac6400 = 58,
    gestaltPerforma6400 = gestaltPowerMac6400,
    gestaltPerforma6360 = gestaltPerforma6400,
    gestaltMacCentris660AV = 60,
    gestaltMacQuadra660AV = gestaltMacCentris660AV,
    gestaltPerforma46x = 62,
    gestaltPowerMac8100_80 = 65,
    gestaltAWS8150_80 = gestaltPowerMac8100_80,
    gestaltPowerMac9500 = 67,
    gestaltPowerMac9600 = gestaltPowerMac9500,
    gestaltPowerMac7500 = 68,
    gestaltPowerMac7600 = gestaltPowerMac7500,
    gestaltPowerMac8500 = 69,
    gestaltPowerMac8600 = gestaltPowerMac8500,
    gestaltAWS8550 = gestaltPowerMac7500,
    gestaltPowerBook180c = 71,
    gestaltPowerBook520 = 72,
    gestaltPowerBook520c = gestaltPowerBook520,
    gestaltPowerBook540 = gestaltPowerBook520,
    gestaltPowerBook540c = gestaltPowerBook520,
    gestaltPowerMac5400 = 74,
    gestaltPowerMac6100_60 = 75,
    gestaltAWS6150_60 = gestaltPowerMac6100_60,
    gestaltPowerBookDuo270c = 77,
    gestaltMacQuadra840AV = 78,
    gestaltPerforma550 = 80,
    gestaltPowerBook165 = 84,
    gestaltPowerBook190 = 85,
    gestaltMacTV = 88,
    gestaltMacLC475 = 89,
    gestaltPerforma47x = gestaltMacLC475,
    gestaltMacLC575 = 92,
    gestaltMacQuadra605 = 94,
    gestaltMacQuadra630 = 98,
    gestaltMacLC580 = 99,
    gestaltPerforma580 = gestaltMacLC580,
    gestaltPowerMac6100_66 = 100,
    gestaltAWS6150_66 = gestaltPowerMac6100_66,
    gestaltPowerBookDuo280 = 102,
    gestaltPowerBookDuo280c = 103,
    gestaltPowerMacLC475 = 104,
    gestaltPowerMacPerforma47x = gestaltPowerMacLC475,
    gestaltPowerMacLC575 = 105,
    gestaltPowerMacPerforma57x = gestaltPowerMacLC575,
    gestaltPowerMacQuadra630 = 106,
    gestaltPowerMacLC630 = gestaltPowerMacQuadra630,
    gestaltPowerMacPerforma63x = gestaltPowerMacQuadra630,
    gestaltPowerMac7200 = 108,
    gestaltPowerMac7300 = 109,
    gestaltPowerMac7100_66 = 112,
    gestaltPowerBook150 = 115,
    gestaltPowerMacQuadra700 = 116,
    gestaltPowerMacQuadra900 = 117,
    gestaltPowerMacQuadra950 = 118,
    gestaltPowerMacCentris610 = 119,
    gestaltPowerMacCentris650 = 120,
    gestaltPowerMacQuadra610 = 121,
    gestaltPowerMacQuadra650 = 122,
```

```
    gestaltPowerMacQuadra800 = 123,
    gestaltPowerBookDuo2300 = 124,
    gestaltPowerBook500PPCUpgrade = 126,
    gestaltPowerBook5300 = 128,
    gestaltPowerBook1400 = 310,
    gestaltPowerBook3400 = 306,
    gestaltPowerBook2400 = 307,
    gestaltPowerBookG3Series = 312,
    gestaltPowerBookG3 = 313,
    gestaltPowerBookG3Series2 = 314,
    gestaltPowerMacNewWorld = 406,
    gestaltPowerMacG3 = 510,
    gestaltPowerMac5500 = 512,
    gestalt20thAnniversary = gestaltPowerMac5500,
    gestaltPowerMac6500 = 513,
    gestaltPowerMac4400_160 = 514,
    gestaltPowerMac4400 = 515,
    gestaltMacOSCompatibility = 1206
};
```

Discussion
To obtain a string containing the machine's name, you can pass the returned value to the `GetIndString` procedure as an index into the resource of type `'STR#'` in the System file having the resource ID defined by the constant `kMachineNameStrID`.

Computer Name Selector

Specifes user-visiblity information for the computer name.

```
enum {
    gestaltUserVisibleMachineName = 'mnam'
};
```

Connection Manager Attribute Selectors

Specify feature availability information for the Connection Manager.

```
enum {
    gestaltConnMgrAttr = 'conn',
    gestaltConnMgrPresent = 0,
    gestaltConnMgrCMSearchFix = 1,
    gestaltConnMgrErrorString = 2,
    gestaltConnMgrMultiAsyncIO = 3
};
```

Constants

`gestaltConnMgrAttr`

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltConnMgrPresent`

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltConnMgrCMSearchFix`
The `gestaltConnMgrCMSearchFix` bit flag indicates that the fix is present that allows the `CMAddSearch` function to work over the `mAttn` channel.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltConnMgrErrorString`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltConnMgrMultiAsyncIO`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Control Manager Attribute Selectors

Specify feature availability information for the Control Manager.

```
enum {
    gestaltControlMgrAttr = 'cntl',
    gestaltControlMgrPresent = (1L << 0),
    gestaltControlMgrPresentBit = 0,
    gestaltControlMsgPresentMask = (1L << gestaltControlMgrPresentBit)
};
```

Constants

`gestaltControlMgrAttr`
The `Gestalt` selector passed to determine what features of the Control Manager are present. This selector is available with Mac OS 8.5 and later. The `Gestalt` function produces a 32-bit value whose bits you should test to determine what Control Manager functionality is available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltControlMgrPresent`
If the bit specified by this mask is set, the Control Manager functionality for Appearance Manager 1.1 is available. This bit is set for Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltControlMgrPresentBit`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltControlMsgPresentMask`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion

Before calling any functions dependent upon the Control Manager, your application should pass the selector `gestaltControlMgrAttr` to the `Gestalt` function to determine which Control Manager functions are available.

Control Manager Version Selector

Specifies version information for the Control Manager.

```
enum {
    gestaltControlMgrVersion = 'cmvr'
};
```

Constants

`gestaltControlMgrVersion`
Available in Mac OS X v10.1 and later.
Declared in `Gestalt.h`.

Control Strip Attribute Selectors

Specify feature availability for the Control Strip.

```
enum {
    gestaltControlStripAttr = 'sdev',
    gestaltControlStripExists = 0,
    gestaltControlStripVersionFixed = 1,
    gestaltControlStripUserFont = 2,
    gestaltControlStripUserHotKey = 3
};
```

Control Strip Version Selector

Specifies version information for the Control Strip.

```
enum {
    gestaltControlStripVersion = 'csvr'
};
```

Constants

`gestaltControlStripVersion`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

CPU Selectors for Apollo

Specify version information for Apollo CPUs.

```
enum {
    gestaltCPUApollo = 0x0111,
    gestaltCPU750FX = 0x0120
};
```

CPU Selectors for Intel and Pentium

Specify version information for Intel and Pentium CPUs.

```
enum {
    gestaltCPU486 = 'i486',
    gestaltCPUPentium = 'i586',
    gestaltCPUPentiumPro = 'i5pr',
    gestaltCPUPentiumII = 'i5ii',
    gestaltCPUX86 = 'ixxx'
};
```

Data Access Manager Attribute Selectors

Specify feature availability information for the Data Access Manager.

```
enum {
    gestaltDBAccessMgrAttr = 'dbac',
    gestaltDBAccessMgrPresent = 0
};
```

Desktop Pictures Attribute Selectors

Specify feature availability information for Desktop Pictures.

```
enum {
    gestaltDesktopPicturesAttr = 'dkpx',
    gestaltDesktopPicturesInstalled = 0,
    gestaltDesktopPicturesDisplayed = 1
};
```

Desktop Printing Attribute Selector

Specify feature availablity information for all desktop printer.

```
enum {
    gestaltDTPInfo = 'dtpx'
};
```

Desktop Printing Driver Attribute Selectors

Specify feature availability for third-party desktop printing drivers.

```
enum {
    gestaltDTPFeatures = 'dtpf',
    kDTPThirdPartySupported = 0x00000004
};
```

Dialog Manager Attribute Selectors

Specify feature availability for the Dialog Manager.

```
enum {
    gestaltDITLExtAttr = 'ditl',
    gestaltDITLExtPresent = 0,
    gestaltDITLExtSupportsIctb = 1
};
```

Constants

`gestaltDITLExtAttr`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltDITLExtPresent`
If this flag bit is `TRUE`, then the Dialog Manager extensions included in System 7 are available.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltDITLExtSupportsIctb`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

Dictionary Manager Attribute Selectors

Specify feature availability information for the Dictionary Manager.

```
enum {
    gestaltDictionaryMgrAttr = 'dict',
    gestaltDictionaryMgrPresent = 0
};
```

Dialog Manager Selectors for Mac OS 8.5

Specify version and feature availability information for the Dialog Manager in Mac OS 8.5.

```
enum {
    gestaltDialogMgrAttr = 'dlog',
    gestaltDialogMgrPresent = (1L << 0),
    gestaltDialogMgrPresentBit = 0,
    gestaltDialogMgrHasAquaAlertBit = 2,
    gestaltDialogMgrPresentMask = (1L << gestaltDialogMgrPresentBit),
    gestaltDialogMgrHasAquaAlertMask = (1L << gestaltDialogMgrHasAquaAlertBit),
    gestaltDialogMsgPresentMask = gestaltDialogMgrPresentMask
};
```

Constants

`gestaltDialogMgrAttr`
The `Gestalt` selector passed to determine what features of the Dialog Manager are present. This selector is available with Mac OS 8.5 and later. Passing `gestaltDialogManagerAttr` produces a 32-bit value whose bits you should test to determine what Dialog Manager functionality is available.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltDialogMgrPresent`
If the bit specified by this mask is set, the Dialog Manager functionality for Appearance Manager 1.1 is available. This bit is set for Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltDialogMgrPresentBit`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltDialogMgrHasAquaAlertBit`
Available in Mac OS X v10.1 and later.

Declared in `Gestalt.h`.

`gestaltDialogMgrPresentMask`
Available in Mac OS X v10.1 and later.

Declared in `Gestalt.h`.

`gestaltDialogMgrHasAquaAlertMask`
Available in Mac OS X v10.1 and later.

Declared in `Gestalt.h`.

`gestaltDialogMsgPresentMask`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion
Before calling any Dialog Manager functions, your application should pass the selector `gestaltDialogManagerAttr` to the `Gestalt` function to determine which Dialog Manager functions are available.

Digital Signature Version Selector

Specifies version information for digital signatures.

```
enum {
    gestaltDigitalSignatureVersion = 'dsig'
};
```

Direct IO Attribute Selector

Specifies availability of direct input/ouput support by the file system.

```
enum {
    gestaltFSSupportsDirectIO = 11
};
```

Disk Cache Size Selector

Specifies size information for the disk cache buffer.

```
enum {
    gestaltDiskCacheSize = 'dcsz'
};
```

Constants

`gestaltDiskCacheSize`

A selector that you pass to the `Gestalt` function. If the function returns `noErr`, the `response` parameter contains the size of the disk cache's buffer. See the Gestalt Manager Reference for more information on the `Gestalt` function.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Display Manager Attribute Selectors

Specify feature availability for the Display Manager.

```
enum {
    gestaltDisplayMgrAttr = 'dply',
    gestaltDisplayMgrPresent = 0,
    gestaltDisplayMgrCanSwitchMirrored = 2,
    gestaltDisplayMgrSetDepthNotifies = 3,
    gestaltDisplayMgrCanConfirm = 4,
    gestaltDisplayMgrColorSyncAware = 5,
    gestaltDisplayMgrGeneratesProfiles = 6,
    gestaltDisplayMgrSleepNotifies = 7
};
```

Constants

`gestaltDisplayMgrAttr`

The `Gestalt` selector you pass to determine which Display Manager attributes are present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltDisplayMgrPresent`

If `true`, the Display Manager is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltDisplayMgrCanSwitchMirrored`

If `true`, the Display Manager can switch modes on mirrored displays.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltDisplayMgrSetDepthNotifies`

If `true`, and you have registered for notification and you will be notified of depth mode changes.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltDisplayMgrCanConfirm`
Not yet supported. Most commonly comes up for display modes that are not marked `kModeSafe`. There is currently no system support for trying an unsafe mode and then restoring if the user does not confirm. When this is supported, this bit will be set.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltDisplayMgrColorSyncAware`
If `true`, Display Manager supports profiles for displays.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltDisplayMgrGeneratesProfiles`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltDisplayMgrSleepNotifies`
Available in Mac OS X v10.2 and later.

Declared in `Gestalt.h`.

Discussion
Before calling any function dependent upon the Display Manager, your application should pass the selector `gestaltDisplayMgrAttr` to the `Gestalt` function to determine the Display Manager attributes that are present.

Display Manager Version Selector

Specifies version information for the Display Manager.

```
enum {
    gestaltDisplayMgrVers = 'dplv'
};
```

Constants
`gestaltDisplayMgrVers`
The `Gestalt` selector you pass to determine what version of the Display Manager is present. For example, a Gestalt result may be 0x00020500, which means that the Display Manager version 2.5 is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion
To determine the version of the current Display Manager, your application should pass the selector `gestaltDisplayMgrVers` to the `Gestalt` function.

Drag Manager Attribute Selectors

Specify feature availability information for the Drag Manager.

```
enum {
    gestaltDragMgrAttr = 'drag',
    gestaltDragMgrPresent = 0,
    gestaltDragMgrFloatingWind = 1,
    gestaltPPCDragLibPresent = 2,
    gestaltDragMgrHasImageSupport = 3,
    gestaltCanStartDragInFloatWindow = 4,
    gestaltSetDragImageUpdates = 5
};
```

Constants

`gestaltDragMgrAttr`
The Gestalt selector passed to determine what features of the Drag Manager are present. Passing the `gestaltDragMgrAttr` constant produces a 32-bit value whose bits you should test to determine what Drag Manager functionality is available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltDragMgrPresent`
If the bit specified by this mask is set, the Drag Manager functions are available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltDragMgrFloatingWind`
If the bit specified by this mask is set, the Drag Manager floating window support functions are available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPPCDragLibPresent`
If the bit specified by this mask is set, the Drag Manager PPC Drag Library functions are available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltDragMgrHasImageSupport`
If the bit specified by this mask is set, the Drag Manager image support functions are available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltCanStartDragInFloatWindow`
If the bit specified by this mask is set, the Drag Manager can start a drag in a floating window.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltSetDragImageUpdates`
Available in Mac OS X v10.1 and later.

Declared in `Gestalt.h`.

Draw Sprocket Version Selectors

Specifies version information for Draw Sprocket.

```
enum {
    gestaltDrawSprocketVersion = 'dspv'
};
```

Easy Access Selectors

Specify version and feature availability information for Easy Access.

```
enum {
    gestaltEasyAccessAttr = 'easy',
    gestaltEasyAccessOff = 0,
    gestaltEasyAccessOn = 1,
    gestaltEasyAccessSticky = 2,
    gestaltEasyAccessLocked = 3
};
```

Edition Manager Attribute Selectors

Specify feature availability for the Edition Manager.

```
enum {
    gestaltEditionMgrAttr = 'edtn',
    gestaltEditionMgrPresent = 0,
    gestaltEditionMgrTranslationAware = 1
};
```

Extension Table Version Selector

Specifies version information for the extension table.

```
enum {
    gestaltExtensionTableVersion = 'etbl'
};
```

File Mapping Attribute Selectors

Specify feature availability for file mapping.

```
enum {
    gestaltFileMappingAttr = 'flmp',
    gestaltFileMappingPresent = 0,
    gestaltFileMappingMultipleFilesFix = 1
};
```

File System Attribute Selectors

Specify feature availability for the file system.

```
enum {
    gestaltFSAttr = 'fs  ',
    gestaltFullExtFSDispatching = 0,
    gestaltHasFSSpecCalls = 1,
    gestaltHasFileSystemManager = 2,
    gestaltFSMDoesDynamicLoad = 3,
    gestaltFSSupports4GBVols = 4,
    gestaltFSSupports2TBVols = 5,
    gestaltHasExtendedDiskInit = 6,
    gestaltDTMgrSupportsFSM = 7,
    gestaltFSNoMFSVols = 8,
    gestaltFSSupportsHFSPlusVols = 9,
    gestaltFSIncompatibleDFA82 = 10
};
```

Constants

`gestaltFSAttr`

A selector you pass to the `Gestalt` function. If the Gestalt function returns `noErr`, the `response` parameter contains a 32-bit value specifying the features of the file system.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltFullExtFSDispatching`

If this bit is set in the `response` parameter, all of the functions selected through the `_HFSDispatch` trap are available to external file systems. If this bit is clear, the File Manager checks the selector passed to `_HFSDispatch` and ensures that it is valid; if the selector is invalid, the result code `paramErr` is returned to the caller. If this bit is set, no such validity checking is performed. See the *Guide to the File System Manager* for more information on external file systems.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasFSSpecCalls`

If this bit is set in the `response` parameter, the operating environment provides the file system specification (`FSSpec`) versions of the basic file-manipulation functions, as well as the `FSMakeFSSpec` function.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasFileSystemManager`

If this bit is set in the `response` parameter, the File System Manager is present. See the *Guide to the File System Manager* for more information about the File System Manager.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltFSMDoesDynamicLoad`

If this bit is set in the `response` parameter, the File System Manager supports dynamic loading of external file system code resources.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltFSSupports4GBVols`

If this bit is set in the `response` parameter, the file system supports 4 gigabyte volumes.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltFSSupports2TBVols`
If this bit is set in the `response` parameter, the file system supports 2 terabyte volumes.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasExtendedDiskInit`
If this bit is set in the `response` parameter, the extended Disk Initialization Package functions are present. These are the `DIXFormat`, `DIXZero`, or `DIReformat` functions. See the *Guide to the File System Manager* for more information about the Disk Initialization Package interfaces.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltDTMgrSupportsFSM`
If this bit is set in the `response` parameter, the desktop database supports File System Manager-based foreign file systems.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltFSNoMFSVols`
If this bit is set in the `response` parameter, the file system does not support MFS volumes.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltFSSupportsHFSPlusVols`
If this bit is set in the `response` parameter, the file system supports HFS Plus volumes.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltFSIncompatibleDFA82`
If this bit is set in the `response` parameter, VCB and FCB structures are changed; DFA 8.2 is incompatible.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

File System Attribute Selectors for Mac OS 9

Specify feature availability for the file system for features introduced in Mac OS 9.

```
enum {
    gestaltHasHFSPlusAPIs = 12,
    gestaltMustUseFCBAccessors = 13,
    gestaltFSUsesPOSIXPathsForConversion = 14,
    gestaltFSSupportsExclusiveLocks = 15,
    gestaltFSSupportsHardLinkDetection = 16
};
```

Constants

`gestaltHasHFSPlusAPIs`
If this bit is set in the `response` parameter, the File Manager supports the HFS Plus APIs. Individual file systems may or may not implement the HFS Plus APIs. However, if this bit is set, the File Manager will emulate the HFS Plus APIs for file systems that do not implement them. Call the functions `PBHGetVolParmsSync` or `PBHGetVolParmsAsync` to determine whether the HFS Plus APIs are directly supported on a given volume.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMustUseFCBAccessors`
If this bit is set in the `response` parameter, the File Manager no longer supports the low memory globals `FCBSPtr` and `FSFCBLen`. All access to file or fork control blocks must use the File System Manager utility functions instead.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltFSUsesPOSIXPathsForConversion`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltFSSupportsExclusiveLocks`
Available in Mac OS X v10.2 and later.

Declared in `Gestalt.h`.

`gestaltFSSupportsHardLinkDetection`
Available in Mac OS X v10.2 and later.

Declared in `Gestalt.h`.

Discussion

Use these constants with the gestalt selector `gestaltFSAttr`, described in "File System Attribute Selectors" (page 1042).

File System Manager Version Selector

Specifies version information for the File System Manager.

```
enum {
    gestaltFSMVersion = 'fsm '
};
```

Constants

`gestaltFSMVersion`

Pass this selector to the `Gestalt` function to determine the version of the HFS External File Systems Manager (FSM).

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

File System Transport Manager Attribute Selectors

Specify feature availability for the File System Transport Manager

```
enum {
    gestaltFXfrMgrAttr = 'fxfr',
    gestaltFXfrMgrPresent = 0,
    gestaltFXfrMgrMultiFile = 1,
    gestaltFXfrMgrErrorString = 2,
    gestaltFXfrMgrAsync = 3
};
```

Constants

`gestaltFXfrMgrAttr`

The selector you pass to the `Gestalt` function to determine the File Transfer Manager attributes.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Find By Content State Selectors

Specify state information for Find By Content.

```
enum {
    gestaltFBCIndexingState = 'fbci',
    gestaltFBCindexingSafe = 0,
    gestaltFBCindexingCritical = 1
};
```

Find By Content Version Selectors

Specify version information for Find By Content.

```
enum {
    gestaltFBCVersion = 'fbcv',
    gestaltFBCCurrentVersion = 0x0011,
    gestaltOSXFBCCurrentVersion = 0x0100
};
```

Find Folder Redirection Attribute Selector

Specifies feature availability information for Find Folder.

```
enum {
    gestaltFindFolderRedirectionAttr = 'fole'
};
```

Finder Attribute Selectors

Specify feature availability for the Finder.

```
enum {
    gestaltFinderAttr = 'fndr',
    gestaltFinderDropEvent = 0,
    gestaltFinderMagicPlacement = 1,
    gestaltFinderCallsAEProcess = 2,
    gestaltOSLCompliantFinder = 3,
    gestaltFinderSupports4GBVolumes = 4,
    gestaltFinderHasClippings = 6,
    gestaltFinderFullDragManagerSupport = 7,
    gestaltFinderFloppyRootComments = 8,
    gestaltFinderLargeAndNotSavedFlavorsOK = 9,
    gestaltFinderUsesExtensibleFolderManager = 10,
    gestaltFinderUnderstandsRedirectedDesktopFolder = 11
};
```

Floppy Driver Attribute Selectors

Specify feature availability information for the floppy disk drive.

```
enum {
    gestaltFloppyAttr = 'flpy',
    gestaltFloppyIsMFMOnly = 0,
    gestaltFloppyIsManualEject = 1,
    gestaltFloppyUsesDiskInPlace = 2
};
```

Font Manager Attribute Selectors

Specify feature availability information for the Font Manager.

```
enum {
    gestaltFontMgrAttr = 'font',
    gestaltOutlineFonts = 0
};
```

Constants

`gestaltFontMgrAttr`
The `Gestalt` selector you pass to determine which Font Manager attributes are present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltOutlineFonts`
If `true`, outline fonts are supported.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion

Before calling any function dependent upon the Font Manager, your application should pass the selector `gestaltDisplayMgrAttr` to the `Gestalt` function to determine the Font Manager attributes that are present.

Folder Manager Attribute Selectors

Specify feature availability information for the Folder Manager.

```
enum {
    gestaltFindFolderAttr = 'fold',
    gestaltFindFolderPresent = 0,
    gestaltFolderDescSupport = 1,
    gestaltFolderMgrFollowsAliasesWhenResolving = 2,
    gestaltFolderMgrSupportsExtendedCalls = 3,
    gestaltFolderMgrSupportsDomains = 4,
    gestaltFolderMgrSupportsFSCalls = 5
};
```

Constants

`gestaltFindFolderAttr`
The selector you pass to the `Gestalt` function to determine the `FindFolder` function attributes.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltFindFolderPresent`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltFolderDescSupport`
If this bit is set, the extended Folder Manager functionality supporting folder descriptors and routings is available. This bit is set for versions of the Mac OS starting with Mac OS 8.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltFolderMgrFollowsAliasesWhenResolving`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltFolderMgrSupportsExtendedCalls`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltFolderMgrSupportsDomains`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltFolderMgrSupportsFSCalls`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion
Before calling any Folder Manager functions, your application should pass the selector `gestaltFindFolderAttr` to the `Gestalt` function to determine which Folder Manager functions are available.

FPU Type Selectors

Specify version and availability information for the type of floating-point unit installed.

```
enum {
    gestaltFPUType = 'fpu ',
    gestaltNoFPU = 0,
    gestalt68881 = 1,
    gestalt68882 = 2,
    gestalt68040FPU = 3
};
```

Constants
`gestaltFPUType`
A constant that represents the type of floating-point unit currently installed, if any.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltNoFPU`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestalt68881`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestalt68882`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestalt68040FPU`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Gestalt Manager Version Selectors

Specify Gestalt Manager version information.

```
enum {
    gestaltVersion = 'vers',
    gestaltValueImplementedVers = 5
};
```

Constants

`gestaltVersion`

The selector you pass to the function `Gestalt` (page 1005) to determine the version of the Gestalt Manager. The function passes back the version in the low-order word of the response.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltValueImplementedVers`

The first version of the Gestalt Manager that implements this selector.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Hardware Attribute Attribute Selectors

Specify feature availability information for hardware.

```
enum {
    gestaltHardwareAttr = 'hdwr',
    gestaltHasVIA1 = 0,
    gestaltHasVIA2 = 1,
    gestaltHasASC = 3,
    gestaltHasSCC = 4,
    gestaltHasSCSI = 7,
    gestaltHasSoftPowerOff = 19,
    gestaltHasSCSI961 = 21,
    gestaltHasSCSI962 = 22,
    gestaltHasUniversalROM = 24,
    gestaltHasEnhancedLtalk = 30
};
```

Constants

`gestaltHardwareAttr`

The selector you pass to the Gestalt function to determine low-level hardware configuration attributes.

Never infer the existence of certain hardware or software features from the responses that `Gestalt` returns when you pass it this selector.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasVIA1`

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasVIA2`

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasASC`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasSCC`
The `gestaltHasSCC` bit is normally returned as 0 on the Macintosh IIfx and Macintosh Quadra 900 computers, which have intelligent I/O processors that isolate the hardware and make direct access to the SCC impossible. However, if the user has used the Compatibility Switch control panel to enable compatibility mode, `gestaltHasSCC` is set.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasSCSI`
The `gestaltHasSCSI` bit means the machine is equipped with a SCSI implementation based on the 53C80 chip, which was introduced in the Macintosh Plus. This bit is 0 on computers with a different SCSI implementation.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasSoftPowerOff`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasSCSI961`
This bit is set if the machine has a SCSI implementation based on the 53C96 chip installed on an internal bus.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasSCSI962`
This bit is set if the machine has a SCSI implementation based on the 53C96 chip installed on an external bus.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasUniversalROM`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasEnhancedLtalk`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Hardware Icon Selector

Specifies icon family resource ID information for the computer hardware.

```
enum {
    gestaltMachineIcon = 'micn'
};
```

Constants

`gestaltMachineIcon`
The selector you pass to the `Gestalt` function to determine the icon family resource ID for the current type of Macintosh.

Never infer the existence of certain hardware or software features from the responses that `Gestalt` returns when you pass it this selector.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Hardware Vendor Selectors

Specify hardware vendor information.

```
enum {
    gestaltHardwareVendorCode = 'hrad',
    gestaltHardwareVendorApple = 'Appl'
};
```

Help Manager Attribute Selectors

Specify feature availability for the Apple Help Manager.

```
enum {
    gestaltHelpMgrAttr = 'help',
    gestaltHelpMgrPresent = 0,
    gestaltHelpMgrExtensions = 1,
    gestaltAppleGuideIsDebug = 30,
    gestaltAppleGuidePresent = 31
};
```

Constants

`gestaltHelpMgrAttr`
The selector you pass to the `Gestalt` function to determine the Help Manager attributes.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHelpMgrPresent`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHelpMgrExtensions`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltAppleGuideIsDebug`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltAppleGuidePresent`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Icon Services Attribute Selectors

Specify feature availability for Icon Services.

```
enum {
    gestaltIconUtilitiesAttr = 'icon',
    gestaltIconUtilitiesPresent = 0,
    gestaltIconUtilitiesHas48PixelIcons = 1,
    gestaltIconUtilitiesHas32BitIcons = 2,
    gestaltIconUtilitiesHas8BitDeepMasks = 3,
    gestaltIconUtilitiesHasIconServices = 4
};
```

Constants

`gestaltIconUtilitiesAttr`
The `Gestalt` selector passed to determine which features of Icon Services are present. The `Gestalt` function produces a 32-bit value whose bits you should test to determine which Icon Services features are available.

Note: available in System 7.0, despite gestalt.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltIconUtilitiesPresent`
True if icon utilities are present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltIconUtilitiesHas48PixelIcons`
True if 48x48 icons are supported by IconUtilities.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltIconUtilitiesHas32BitIcons`
True if 32-bit deep icons are supported.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltIconUtilitiesHas8BitDeepMasks`
True if 8-bit deep masks are supported.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltIconUtilitiesHasIconServices`
True if IconServices is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion
Before calling any Icon Services functions, your application should pass the selector `gestaltIconUtilitiesAttr` to the `Gestalt` function.

Image Compression Manager Version Selector

Specifies the version of the Image Compression Manager.

```
enum {
    gestaltCompressionMgr = 'icmp'
};
```

Constants
`gestaltCompressionMgr`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

Intel Architecture Selector

Specifies the Intel architecture.

```
enum {
    gestaltIntel = 10
};
```

Internal Display Location Selector

Specifies the slot number information for the internal display location.

```
enum {
    gestaltInternalDisplay = 'idsp'
};
```

Keyboard Selectors

Specify keyboard information.

```
enum {
    gestaltKeyboardType = 'kbd ',
    gestaltMacKbd = 1,
    gestaltMacAndPad = 2,
    gestaltMacPlusKbd = 3,
    gestaltExtADBKbd = 4,
    gestaltStdADBKbd = 5,
    gestaltPrtblADBKbd = 6,
    gestaltPrtblISOKbd = 7,
    gestaltStdISOADBKbd = 8,
    gestaltExtISOADBKbd = 9,
    gestaltADBKbdII = 10,
    gestaltADBISOKbdII = 11,
    gestaltPwrBookADBKbd = 12,
    gestaltPwrBookISOADBKbd = 13,
    gestaltAppleAdjustKeypad = 14,
    gestaltAppleAdjustADBKbd = 15,
    gestaltAppleAdjustISOKbd = 16,
    gestaltJapanAdjustADBKbd = 17,
    gestaltPwrBkExtISOKbd = 20,
    gestaltPwrBkExtJISKbd = 21,
    gestaltPwrBkExtADBKbd = 24,
    gestaltPS2Keyboard = 27,
    gestaltPwrBkSubDomKbd = 28,
    gestaltPwrBkSubISOKbd = 29,
    gestaltPwrBkSubJISKbd = 30,
    gestaltPwrBkEKDomKbd = 195,
    gestaltPwrBkEKISOKbd = 196,
    gestaltPwrBkEKJISKbd = 197,
    gestaltUSBCosmoANSIKbd = 198,
    gestaltUSBCosmoISOKbd = 199,
    gestaltUSBCosmoJISKbd = 200,
    gestaltPwrBk99JISKbd = 201,
    gestaltUSBAndyANSIKbd = 204,
    gestaltUSBAndyISOKbd = 205,
    gestaltUSBAndyJISKbd = 206
};
```

Constants

`gestaltKeyboardType`

The selector you pass to the `Gestalt` function to determine the type of the keyboard.

If the Apple Desktop Bus (ADB) is in use, there may be multiple keyboards or other ADB devices attached to the machine. The `gestaltKeyboardType` selector identifies only the type of the keyboard on which the last keystroke occurred.

You cannot use this selector to find out what ADB devices are connected. For that, you can use the Apple Desktop Bus Manager. Note that the ADB keyboard types described by `Gestalt` do not necessarily map directly to ADB device handler IDs.

Future support for the `gestaltKeyboardType` selector is not guaranteed. To determine the type of the keyboard last touched without using `Gestalt`, check the system global variable `KbdType`.

If the Gestalt Manager does not recognize the keyboard type, it returns an error.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Keyboard Selectors for Laptops

Specify laptop keyboard information.

```
enum {
    gestaltPortable2001ANSIKbd = 202,
    gestaltPortable2001ISOKbd = 203,
    gestaltPortable2001JISKbd = 207
};
```

Logical Page Size Selector

Specifies logical page size information.

```
enum {
    gestaltLogicalPageSize = 'pgsz'
};
```

Constants

`gestaltLogicalPageSize`

The logical page size. This value is defined only on machines with the MC68010, MC68020, MC68030, or MC68040 microprocessors. On a machine with the MC68000, the `Gestalt` function returns an error when called with this selector.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Logical RAM Size Selector

Specifies logical randon-access memory size information.

```
enum {
    gestaltLogicalRAMSize = 'lram'
};
```

Constants

`gestaltLogicalRAMSize`

The amount of logical memory available. This value is the same as that returned by `gestaltPhysicalRAMSize` when virtual memory is not installed. On some machines, however, this value might be less than the value returned by `gestaltPhysicalRAMSize` because some RAM may be used by the video display and the Operating System.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Low Memory Size Selector

Specifies information about the size of the low-memory area.

```
enum {
    gestaltLowMemorySize = 'lmem'
};
```

Constants

`gestaltLowMemorySize`

The size (in bytes) of the low-memory area. The low-memory area is used for vectors, global variables, and dispatch tables

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Machine Name String ID

Defines a machine name string ID.

```
enum {
    kMachineNameStrID = -16395
};
```

Mailer Version Selector

Specifies version information for the OCE standard mailer.

```
enum {
    gestaltSMPMailerVersion = 'malr'
};
```

Mailer Send LetterVersion Selector

Specifies version information for the OCE standard mailer's send letter.

```
enum {
    gestaltSMPSPSendLetterVersion = 'spsl'
};
```

Media Bay Selectors

Specify information about media bay availability.

```
enum {
    gestaltMediaBay = 'mbeh',
    gestaltMBLegacy = 0,
    gestaltMBSingleBay = 1,
    gestaltMBMultipleBays = 2
};
```

Memory Attribute Selectors

Specify feature availability information for memory.

```
enum {
    gestaltOSAttr = 'os  ',
    gestaltSysZoneGrowable = 0,
    gestaltLaunchCanReturn = 1,
    gestaltLaunchFullFileSpec = 2,
    gestaltLaunchControl = 3,
    gestaltTempMemSupport = 4,
    gestaltRealTempMemory = 5,
    gestaltTempMemTracked = 6,
    gestaltIPCSupport = 7,
    gestaltSysDebuggerSupport = 8,
    gestaltNativeProcessMgrBit = 19,
    gestaltAltivecRegistersSwappedCorrectlyBit = 20
};
```

Constants

`gestaltOSAttr`

The `Gestalt` selector you pass to determine general Operating System attributes, such as whether temporary memory handles are real handles. The low-order bits of the response parameter are interpreted as bit flags. A flag is set to 1 to indicate that the corresponding feature is available. Currently, the following bits are significant.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltSysZoneGrowable`

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltLaunchCanReturn`

If this bit is set, the `_Launch` trap macro can return to the caller. The `_Launch` trap macro in system software version 7.0 (and in earlier versions running MultiFinder) gives your application the option to continue running after it launches another application. In earlier versions of system software not running MultiFinder, the `_Launch` trap macro forces the launching application to quit.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltLaunchFullFileSpec`

If this bit is set, the `launchControlFlags` field supports control flags in addition to the `launchContinue` flag, and if the `_Launch` trap can process the `launchAppSpec`, `launchProcessSN`, `launchPreferredSize`, `launchMinimumSize`, `launchAvailableSize`, and `launchAppParameters` fields in the launch parameter block.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltLaunchControl`

If this bit is set, the Process Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltTempMemSupport`

If `true`, there is temporary memory support.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltRealTempMemory`
If `true`, temporary memory handles are real.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltTempMemTracked`
If `true`, temporary memory handles are tracked.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltIPCSupport`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltSysDebuggerSupport`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltNativeProcessMgrBit`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltAltivecRegistersSwappedCorrectlyBit`
Available in Mac OS X v10.2 and later.
Declared in `Gestalt.h`.

Memory Mapping Attribute Selectors

Specify feature availibility information for memory mapping.

```
enum {
    gestaltMemoryMapAttr = 'mmap',
    gestaltMemoryMapSparse = 0
};
```

Menu Manager Selectors in Mac OS 8.5

Specify version and feature availability information for the Menu Manager in Mac OS 8.5

```
enum {
    gestaltMenuMgrAttr = 'menu',
    gestaltMenuMgrPresent = (1L << 0),
    gestaltMenuMgrPresentBit = 0,
    gestaltMenuMgrAquaLayoutBit = 1,
    gestaltMenuMgrMultipleItemsWithCommandIDBit = 2,
    gestaltMenuMgrRetainsIconRefBit = 3,
    gestaltMenuMgrSendsMenuBoundsToDefProcBit = 4,
    gestaltMenuMgrMoreThanFiveMenusDeepBit = 5,
    gestaltMenuMgrPresentMask = (1L << gestaltMenuMgrPresentBit),
    gestaltMenuMgrAquaLayoutMask = (1L << gestaltMenuMgrAquaLayoutBit),
    gestaltMenuMgrMultipleItemsWithCommandIDMask = (1L <<
gestaltMenuMgrMultipleItemsWithCommandIDBit),
    gestaltMenuMgrRetainsIconRefMask = (1L << gestaltMenuMgrRetainsIconRefBit),
    gestaltMenuMgrSendsMenuBoundsToDefProcMask = (1L <<
gestaltMenuMgrSendsMenuBoundsToDefProcBit),
    gestaltMenuMgrMoreThanFiveMenusDeepMask = (1L <<
gestaltMenuMgrMoreThanFiveMenusDeepBit)
};
```

Constants

`gestaltMenuMgrAttr`
The `Gestalt` selector passed to determine what features of the Menu Manager are present. This selector is available with Mac OS 8.5 and later. Passing `gestaltMenuMgrAttr` produces a 32-bit value whose bits you should test to determine what Menu Manager functionality is available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMenuMgrPresent`
If the bit specified by this mask is set, the Menu Manager functionality for Appearance Manager 1.1 is available. This bit is set for Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMenuMgrPresentBit`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMenuMgrAquaLayoutBit`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMenuMgrMultipleItemsWithCommandIDBit`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMenuMgrRetainsIconRefBit`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMenuMgrSendsMenuBoundsToDefProcBit`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMenuMgrMoreThanFiveMenusDeepBit`
Available in Mac OS X v10.2 and later.

Declared in `Gestalt.h`.

`gestaltMenuMgrPresentMask`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMenuMgrAquaLayoutMask`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMenuMgrMultipleItemsWithCommandIDMask`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMenuMgrRetainsIconRefMask`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMenuMgrSendsMenuBoundsToDefProcMask`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMenuMgrMoreThanFiveMenusDeepMask`
Available in Mac OS X v10.2 and later.

Declared in `Gestalt.h`.

Discussion
Before calling any Menu Manager functions, your application should pass the selector `gestaltMenuMgrAttr` to the `Gestalt` function to determine which Menu Manager functions are available.

Message Manager Version Selector

Specify version information for the Message Manager.

```
enum {
    gestaltMessageMgrVersion = 'mess'
};
```

Miscellaneous Attribute Selectors

Specify feature availability information for miscellaneous pieces of the operating system or the hardware configuration.

```
enum {
    gestaltMiscAttr = 'misc',
    gestaltScrollingThrottle = 0,
    gestaltSquareMenuBar = 2
};
```

Constants

`gestaltMiscAttr`
The selector you pass to the `Gestalt` function to determine information about miscellaneous pieces of the Operating System or hardware configuration.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltScrollingThrottle`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltSquareMenuBar`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Mixed Mode Manager Selectors

Specify version and feature availability information for the Mixed Mode Manager.

```
enum {
    gestaltMixedModeAttr = 'mixd',
    gestaltMixedModePowerPC = 0,
    gestaltPowerPCAware = 0,
    gestaltMixedModeCFM68K = 1,
    gestaltMixedModeCFM68KHasTrap = 2,
    gestaltMixedModeCFM68KHasState = 3
};
```

Constants

`gestaltMixedModeAttr`
The Gestalt selector you pass to determine what version of Mixed Mode Manager is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMixedModePowerPC`
True if Mixed Mode supports PowerPC ABI calling conventions

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPowerPCAware`
Old name for `gestaltMixedModePowerPC`

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMixedModeCFM68K`
: True if Mixed Mode supports CFM-68K calling conventions

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMixedModeCFM68KHasTrap`
: True if CFM-68K Mixed Mode implements `_MixedModeDispatch` (versions 1.0.1 and prior did not)

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMixedModeCFM68KHasState`
: True if CFM-68K Mixed Mode exports Save/RestoreMixedModeState

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion
Before calling any function dependent upon Mixed Mode Manager, your application should pass the selector `gestaltMixedModeAttr` to the `Gestalt` function to determine the Mixed Mode Manager attributes that are present.

Mixed Mode Manager Version Selector

Specifies version information for the Mixed Mode Manager.

```
enum {
    gestaltMixedModeVersion = 'mixd'
};
```

Constants
`gestaltMixedModeVersion`
: The selector you pass to the `Gestalt` function to determine the version of Mixed Mode Manager.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

MMU Type Selectors

Specify information about the type of MMU installed.

```
enum {
    gestaltMMUType = 'mmu ',
    gestaltNoMMU = 0,
    gestaltAMU = 1,
    gestalt68851 = 2,
    gestalt68030MMU = 3,
    gestalt68040MMU = 4,
    gestaltEMMU1 = 5
};
```

Constants

`gestaltMMUType`
The selector you pass to the `Gestalt` function to determine the type of MMU currently installed.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltNoMMU`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltAMU`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestalt68851`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestalt68030MMU`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestalt68040MMU`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltEMMU1`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

Multiple Users State Selector

Specifies information about the multiple user state.

```
enum {
    gestaltMultipleUsersState = 'mfdr'
};
```

Name-Binding Protocol Attribute Selectors

Specify feature availiability information for the standard name-binding protocol.

```
enum {
    gestaltStdNBPAttr = 'nlup',
    gestaltStdNBPPresent = 0,
    gestaltStdNBPSupportsAutoPosition = 1
};
```

Constants

`gestaltStdNBPAttr`
The selector you pass to the `Gestalt` function to determine information about the StandardNBP (Name-Binding Protocol) function.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltStdNBPPresent`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltStdNBPSupportsAutoPosition`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Name Registry Version Selector

Specifies the version of the name registry.

```
enum {
    gestaltNameRegistryVersion = 'nreg'
};
```

Constants

`gestaltNameRegistryVersion`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Native CPU Selectors

Specify the native CPU type or family.

```
enum {
    gestaltNativeCPUtype = 'cput',
    gestaltNativeCPUfamily = 'cpuf',
    gestaltCPU68000 = 0,
    gestaltCPU68010 = 1,
    gestaltCPU68020 = 2,
    gestaltCPU68030 = 3,
    gestaltCPU68040 = 4,
    gestaltCPU601 = 0x0101,
    gestaltCPU603 = 0x0103,
    gestaltCPU604 = 0x0104,
    gestaltCPU603e = 0x0106,
    gestaltCPU603ev = 0x0107,
    gestaltCPU750 = 0x0108,
    gestaltCPU604e = 0x0109,
    gestaltCPU604ev = 0x010A,
    gestaltCPUG4 = 0x010C,
    gestaltCPUG47450 = 0x0110
};
```

Constants

`gestaltNativeCPUtype`
The selector you pass to the `Gestalt` function to determine the native CPU type.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltNativeCPUfamily`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltCPU68000`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltCPU68010`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltCPU68020`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltCPU68030`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltCPU68040`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltCPU601`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltCPU603`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltCPU604`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltCPU603e`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltCPU603ev`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltCPU750`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltCPU604e`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltCPU604ev`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltCPUG4`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltCPUG47450`
Available in Mac OS X v10.2 and later.
Declared in `Gestalt.h`.

Discussion
The use of these selectors is no longer recommended. You can use the `gestaltSysArchitecture` (page 1098) selector to determine whether your application is running on a PowerPC or Intel-based Macintosh. If you are trying to determine whether you can use a particular processor feature, you should check directly for that feature using a BSD library function such as `sysctl` or `sysctlbyname`. For more information, see *Mac OS X Man Pages*.

Notification Manager Attribute Selectors

Specify feature availability information for the Notification Manager.

```
enum {
    gestaltNotificationMgrAttr = 'nmgr',
    gestaltNotificationPresent = 0
};
```

Constants
`gestaltNotificationMgrAttr`
.The Gestalt selector which you pass to the `Gestalt` function to determine Notification Manager attributes.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltNotificationPresent`
True if the Notification Manager exists.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

NuBus Location Selector

Specifies information about the NuBus slot connector locations.

```
enum {
    gestaltNuBusConnectors = 'sltc'
};
```

Constants

`gestaltNuBusConnectors`
A bitmap that describes the NuBus slot connector locations. On a Macintosh II, for example, the return value would have bits 9 through 14 set, indicating that 6 NuBus slots are present, at locations 9 through 14.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

NuBus Slot Count Selector

Specifies information about the number of NuBus slots.

```
enum {
    gestaltNuBusSlotCount = 'nubs'
};
```

OCE Toolbox Attribute Selectors

Specify feature availability for the OCE Toolbox.

```
enum {
    gestaltOCEToolboxAttr = 'oceu',
    gestaltOCETBPresent = 0x01,
    gestaltOCETBAvailable = 0x02,
    gestaltOCESFServerAvailable = 0x04,
    gestaltOCETBNativeGlueAvailable = 0x10
};
```

OCE Toolbox Version Selectors

Specify version information for the OCE Toolbox.

```
enum {
    gestaltOCEToolboxVersion = 'ocet',
    gestaltOCETB = 0x0102,
    gestaltSFServer = 0x0100
};
```

Open Firmware Selector

Specifies version information for Open Firmware.

```
enum {
    gestaltOpenFirmwareInfo = 'opfw'
};
```

Open Firmware Safe Selectors

Specify feature availability for Open Firmware safe features.

```
enum {
    gestaltSafeOFAttr = 'safe',
    gestaltVMZerosPagesBit = 0,
    gestaltInitHeapZerosOutHeapsBit = 1,
    gestaltNewHandleReturnsZeroedMemoryBit = 2,
    gestaltNewPtrReturnsZeroedMemoryBit = 3,
    gestaltFileAllocationZeroedBlocksBit = 4
};
```

Open Transport Selectors

Specify version and feature availability information for Open Transport.

```
enum {
    gestaltOpenTpt = 'otan',
    gestaltOpenTptPresentMask = 0x00000001,
    gestaltOpenTptLoadedMask = 0x00000002,
    gestaltOpenTptAppleTalkPresentMask = 0x00000004,
    gestaltOpenTptAppleTalkLoadedMask = 0x00000008,
    gestaltOpenTptTCPPresentMask = 0x00000010,
    gestaltOpenTptTCPLoadedMask = 0x00000020,
    gestaltOpenTptIPXSPXPresentMask = 0x00000040,
    gestaltOpenTptIPXSPXLoadedMask = 0x00000080,
    gestaltOpenTptPresentBit = 0,
    gestaltOpenTptLoadedBit = 1,
    gestaltOpenTptAppleTalkPresentBit = 2,
    gestaltOpenTptAppleTalkLoadedBit = 3,
    gestaltOpenTptTCPPresentBit = 4,
    gestaltOpenTptTCPLoadedBit = 5,
    gestaltOpenTptIPXSPXPresentBit = 6,
    gestaltOpenTptIPXSPXLoadedBit = 7
};
```

Open Transport Network Setup Selectors

Specify feature availability and setup information for Open Transport networking.

```
enum {
    gestaltOpenTptNetworkSetup = 'otcf',
    gestaltOpenTptNetworkSetupLegacyImport = 0,
    gestaltOpenTptNetworkSetupLegacyExport = 1,
    gestaltOpenTptNetworkSetupSupportsMultihoming = 2
};
```

Open Transport Network Version Selector

Specifies the version of the Open Transport network setup.

```
enum {
    gestaltOpenTptNetworkSetupVersion = 'otcv'
};
```

Open Transport Remote Access Selectors

Specify feature availabiliy for Open Transport remote access.

```
enum {
    gestaltOpenTptRemoteAccess = 'otra',
    gestaltOpenTptRemoteAccessPresent = 0,
    gestaltOpenTptRemoteAccessLoaded = 1,
    gestaltOpenTptRemoteAccessClientOnly = 2,
    gestaltOpenTptRemoteAccessPServer = 3,
    gestaltOpenTptRemoteAccessMPServer = 4,
    gestaltOpenTptPPPPresent = 5,
    gestaltOpenTptARAPPresent = 6
};
```

Opent Transport Remote Access Version Selector

Specifies version information for Open Transport remote access.

```
enum {
    gestaltOpenTptRemoteAccessVersion = 'otrv'
};
```

Open Transport Version Selector

Specifies version information for Open Transport.

```
enum {
    gestaltOpenTptVersions = 'otvr'
};
```

OS Trap Table Selector

Specifies base address information for the operating system trap dispatch table.

```
enum {
    gestaltOSTable = 'ostt'
};
```

Constants

`gestaltOSTable`

The selector you pass to the `Gestalt` function to determine the base address of the operating system trap dispatch table.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Parity Checking Attribute Selectors

Specify feature availability for parity checking.

```
enum {
    gestaltParityAttr = 'prty',
    gestaltHasParityCapability = 0,
    gestaltParityEnabled = 1
};
```

Constants

`gestaltParityAttr`
The selector you pass to the `Gestalt` function to determine information about the machine's parity-checking features.

Note that parity is not considered to be enabled unless all installed memory is parity RAM.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasParityCapability`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltParityEnabled`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

PC Compatibility Card Selectors

Specify version and feature availability information for a PC-compatibility card.

```
enum {
    gestaltPCCard = 'pccd',
    gestaltCardServicesPresent = 0,
    gestaltPCCardFamilyPresent = 1,
    gestaltPCCardHasPowerControl = 2,
    gestaltPCCardSupportsCardBus = 3
};
```

PC Exchange Attribute Selectors

Specify feature availability information for PC Exchange.

```
enum {
    gestaltPCXAttr = 'pcxg',
    gestaltPCXHas8and16BitFAT = 0,
    gestaltPCXHasProDOS = 1,
    gestaltPCXNewUI = 2,
    gestaltPCXUseICMapping = 3
};
```

Constants

`gestaltPCXAttr`
The selector you pass to the `Gestalt` function to determine the PC Exchange attributes.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPCXHas8and16BitFAT`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPCXHasProDOS`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPCXNewUI`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPCXUseICMapping`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Physical RAM Size Selector

Specifies information about the size of the physical RAM.

```
enum {
    gestaltPhysicalRAMSize = 'ram '
};
```

Constants

`gestaltPhysicalRAMSize`
The selector you pass to the `Gestalt` function to determine the number of bytes of physical RAM currently installed.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Pop-up Control Selector

Specify feature availability for pop-up controls.

```
enum {
    gestaltPopupAttr = 'pop!',
    gestaltPopupPresent = 0
};
```

Constants

`gestaltPopupAttr`
The selector you pass to the `Gestalt` function to determine the attribute of the pop-up control definition.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPopupPresent`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Power Manager Attribute Selectors

Specify feature availability for the Power Manager.

```
enum {
    gestaltPowerMgrAttr = 'powr',
    gestaltPMgrExists = 0,
    gestaltPMgrCPUIdle = 1,
    gestaltPMgrSCC = 2,
    gestaltPMgrSound = 3,
    gestaltPMgrDispatchExists = 4,
    gestaltPMgrSupportsAVPowerStateAtSleepWake = 5
};
```

Constants

`gestaltPowerMgrAttr`
The `Gestalt` selector you pass to determine which Power Manager capabilities are available.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltPMgrExists`
If true, the Power Manager is present.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltPMgrCPUIdle`
If true the CPU is capable of going into a low–power-consumption state.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltPMgrSCC`
If true, it is possible to stop the SCC clock, thus effectively turning off the serial ports.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltPMgrSound`
If true, it is possible to turn off power to the sound circuits.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltPMgrDispatchExists`
If true, Dispatch is present.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltPMgrSupportsAVPowerStateAtSleepWake`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

Power Manager Version Selector

Specifies version information for the Power Manager.

```
enum {
    gestaltPowerMgrVers = 'pwrv'
};
```

PowerPC Attribute Selectors

Specify feature availability for PowerPC processors.

```
enum {
    gestaltPowerPCProcessorFeatures = 'ppcf',
    gestaltPowerPCHasGraphicsInstructions = 0,
    gestaltPowerPCHasSTFIWXInstruction = 1,
    gestaltPowerPCHasSquareRootInstructions = 2,
    gestaltPowerPCHasDCBAInstruction = 3,
    gestaltPowerPCHasVectorInstructions = 4,
    gestaltPowerPCHasDataStreams = 5
};
```

PowerPC Toolbox Attribute Selectors

Specify feature availability for the PowerPC Toolbox.

```
enum {
    gestaltPPCToolboxAttr = 'ppc ',
    gestaltPPCToolboxPresent = 0x0000,
    gestaltPPCSupportsRealTime = 0x1000,
    gestaltPPCSupportsIncoming = 0x0001,
    gestaltPPCSupportsOutGoing = 0x0002,
    gestaltPPCSupportsTCP_IP = 0x0004,
    gestaltPPCSupportsIncomingAppleTalk = 0x0010,
    gestaltPPCSupportsIncomingTCP_IP = 0x0020,
    gestaltPPCSupportsOutgoingAppleTalk = 0x0100,
    gestaltPPCSupportsOutgoingTCP_IP = 0x0200
};
```

Constants

`gestaltPPCToolboxAttr`
The selector you pass to the Gestalt function to determine the Program-to-Program Communication (PPC) Toolbox attributes. Note that these constants are defined as masks, not bit numbers.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPPCToolboxPresent`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPPCSupportsRealTime`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPPCSupportsIncoming`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPPCSupportsOutGoing`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPPCSupportsTCP_IP`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPPCSupportsIncomingAppleTalk`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPPCSupportsIncomingTCP_IP`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPPCSupportsOutgoingAppleTalk`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPPCSupportsOutgoingTCP_IP`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Preemptive Function Atrribute Selectors

Specify feature availability information for preemptive system software functions.

```
enum {
    gestaltMPCallableAPIsAttr = 'mpsc',
    gestaltMPFileManager = 0,
    gestaltMPDeviceManager = 1,
    gestaltMPTrapCalls = 2
};
```

Constants

`gestaltMPCallableAPIsAttr`
The Gestalt selector passed to determine the availability of preemptive system software functions. The `Gestalt` function produces a 32-bit value that you should test to determine which what type of preemptive calls are allowed.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMPFileManager`
If this bit is set, you can call preemptively safe File Manager functions.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMPDeviceManager`
If this bit is set, you can call preemptively safe Device Manager function.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMPTrapCalls`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion
Before calling any Mac OS system software functions from a preemptive task , you should call the `Gestalt` function with the `gestaltMPCallableAPIsAttr` selector set to determine which preemptively safe system calls are allowed.

Note that for functions that are shared between managers (for example, `PBCloseSync`), you should check the bit that is appropriate for the manager you want to call.

Version Notes
Introduced with Multiprocessing Services 2.1

Processor Clock Speed Selector

Specifies information about processor clock speed.

```
enum {
    gestaltProcClkSpeed = 'pclk'
};
```

Processor Type Selector

Specifies information about the type of microprocessor.

```
enum {
    gestaltProcessorType = 'proc',
    gestalt68000 = 1,
    gestalt68010 = 2,
    gestalt68020 = 3,
    gestalt68030 = 4,
    gestalt68040 = 5
};
```

Constants
`gestaltProcessorType`
The selector you pass to the `Gestalt` function to determine the type of microprocessor currently running.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestalt68000`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestalt68010`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestalt68020`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestalt68030`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestalt68040`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

Quadra Redefinitions

Specifies alternate names for MacQuadra constants.

```
enum {
    gestaltQuadra605 = gestaltMacQuadra605,
    gestaltQuadra610 = gestaltMacQuadra610,
    gestaltQuadra630 = gestaltMacQuadra630,
    gestaltQuadra650 = gestaltMacQuadra650,
    gestaltQuadra660AV = gestaltMacQuadra660AV,
    gestaltQuadra700 = gestaltMacQuadra700,
    gestaltQuadra800 = gestaltMacQuadra800,
    gestaltQuadra840AV = gestaltMacQuadra840AV,
    gestaltQuadra900 = gestaltMacQuadra900,
    gestaltQuadra950 = gestaltMacQuadra950
};
```

QuickDraw 3D Attribute Selectors

Specify feature availability information for QuickDraw 3D.

```
enum {
    gestaltQD3D = 'qd3d',
    gestaltQD3DPresent = 0
};
```

Quick Draw 3D Old Attribute Selectors

Specify old feature availability information for QuickDraw 3D.

```
enum {
    gestaltQD3DNotPresent = (0 << gestaltQD3DPresent),
    gestaltQD3DAvailable = (1 << gestaltQD3DPresent)
};
```

Quick Draw 3D Version Selector

Specifies version information for QuickDraw 3D.

```
enum {
    gestaltQD3DVersion = 'q3v '
};
```

QuickDraw 3D Viewer Attribute Selectors

Specify feature availablity information for QuickDraw 3D Viewer.

```
enum {
    gestaltQD3DViewer = 'q3vc',
    gestaltQD3DViewerPresent = 0
};
```

QuickDraw Attribute Selectors

Specify feature availability information for QuickDraw.

```
enum {
    gestaltQuickdrawFeatures = 'qdrw',
    gestaltHasColor = 0,
    gestaltHasDeepGWorlds = 1,
    gestaltHasDirectPixMaps = 2,
    gestaltHasGrayishTextOr = 3,
    gestaltSupportsMirroring = 4,
    gestaltQDHasLongRowBytes = 5
};
```

Constants

`gestaltQuickdrawFeatures`
The selector you pass to the `Gestalt` function to determine the QuickDraw features.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasColor`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasDeepGWorlds`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasDirectPixMaps`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasGrayishTextOr`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltSupportsMirroring`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltQDHasLongRowBytes`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

QuickDraw Version Selectors

Specify version information for QuickDraw.

```
enum {
    gestaltQuickdrawVersion = 'qd  ',
    gestaltOriginalQD = 0x0000,
    gestalt8BitQD = 0x0100,
    gestalt32BitQD = 0x0200,
    gestalt32BitQD11 = 0x0201,
    gestalt32BitQD12 = 0x0220,
    gestalt32BitQD13 = 0x0230,
    gestaltAllegroQD = 0x0250,
    gestaltMacOSXQD = 0x0300
};
```

Constants

`gestaltQuickdrawVersion`
The Gestalt selector you pass to determine what version of QuickDraw is present. For QuickDraw Text, the Gestalt selector you pass to determine what version of QuickDraw Text is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltOriginalQD`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestalt8BitQD`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestalt32BitQD`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestalt32BitQD11`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestalt32BitQD12`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestalt32BitQD13`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltAllegroQD`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMacOSXQD`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion
The version of QuickDraw is encoded as a revision number in the low-order word of the return value. The high-order byte represents the major revision number, and the low-order byte represents the minor revision number. For example, version 1.3 of 32-Bit QuickDraw represents QuickDraw revision 2.3; its response value is $0230.

Values having a major revision number of 1 or 2 indicate that Color QuickDraw is available, in either the 8-bit or 32-bit version. These results do not, however, indicate whether a color monitor is attached to the system. You must use high-level QuickDraw functions to obtain that information.

QuickDraw GX Overall Version Selector

Specifies version information for the overall version of QuickDraw GX.

```
enum {
    gestaltGXVersion = 'qdgx'
};
```

QuickDraw GX Printing Version Selector

Specifies version information for QuickDraw GX printing.

```
enum {
    gestaltGXPrintingMgrVersion = 'pmgr'
};
```

QuickDraw GX Version Selectors

Specify version information for QuickDraw GX.

```
enum {
    gestaltGraphicsVersion = 'grfx',
    gestaltCurrentGraphicsVersion = 0x00010200
};
```

Constants
`gestaltGraphicsVersion`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltCurrentGraphicsVersion`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

QuickDraw GX Attribute Selectors

Specify feature availability information for QuickDraw GX.

```
enum {
    gestaltGraphicsAttr = 'gfxa',
    gestaltGraphicsIsDebugging = 0x00000001,
    gestaltGraphicsIsLoaded = 0x00000002,
    gestaltGraphicsIsPowerPC = 0x00000004
};
```

QuickDraw 3D Viewer Old Selectors

Specify old feature availability information for QuickDraw 3D.

```
enum {
    gestaltQD3DViewerNotPresent = (0 << gestaltQD3DViewerPresent),
    gestaltQD3DViewerAvailable = (1 << gestaltQD3DViewerPresent)
};
```

QuickDraw Text Attribute Selectors

Specify feature availability information for QuickDraw Text.

```
enum {
    gestaltQDTextFeatures = 'qdtf',
    gestaltWSIISupport = 0,
    gestaltSbitFontSupport = 1,
    gestaltAntiAliasedTextAvailable = 2,
    gestaltOFA2available = 3,
    gestaltCreatesAliasFontRsrc = 4,
    gestaltNativeType1FontSupport = 5,
    gestaltCanUseCGTextRendering = 6
};
```

Constants

`gestaltQDTextFeatures`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltWSIISupport`
WSII support is included.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltSbitFontSupport`
sbit-only fonts are supported.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltAntiAliasedTextAvailable`
Capable of antialiased text.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltOFA2available`
OFA2 is available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltCreatesAliasFontRsrc`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltNativeType1FontSupport`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltCanUseCGTextRendering`
Available in Mac OS X v10.2 and later.

Declared in `Gestalt.h`.

Discussion
Before calling any function dependent upon QuickDraw Text, your application should pass the selector `gestaltQDTextFeatures` to the Gestalt function to determine the QuickDraw Text attributes that are present.

QuickDraw Text Version Selectors

Specify version information for QuickDraw Text.

```
enum {
    gestaltQDTextVersion = 'qdtx',
    gestaltOriginalQDText = 0x0000,
    gestaltAllegroQDText = 0x0100,
    gestaltMacOSXQDText = 0x0200
};
```

Constants
`gestaltQDTextVersion`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltOriginalQDText`
This is the original version of QuickDraw Text, used through Mac OS 8.1.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltAllegroQDText`
This is the version of QuickDraw Text used with Mac OS 8.2 and up.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMacOSXQDText`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion
To determine the version of the current QuickDraw Text, your application should pass the selector `gestaltQuickdrawVersion` to the Gestalt function.

QuickTime VR Feature Selectors

Specify feature availability information for QuickTime VR.

```
enum {
    gestaltQTVRMgrAttr = 'qtvr',
    gestaltQTVRMgrPresent = 0,
    gestaltQTVRObjMoviesPresent = 1,
    gestaltQTVRCylinderPanosPresent = 2,
    gestaltQTVRCubicPanosPresent = 3
};
```

QuickTime VR Version Selector

Specifies version information for QuickTime VR.

```
enum {
    gestaltQTVRMgrVers = 'qtvv'
};
```

QuickTime Attribute Selectors

Specify feature availability information for QuickTime.

```
enum {
    gestaltQuickTimeFeatures = 'qtrs',
    gestaltPPCQuickTimeLibPresent = 0
};
```

QuickTime Version Selectors

Specify version information for QuickTime.

```
enum {
    gestaltQuickTimeVersion = 'qtim',
    gestaltQuickTime = 'qtim'
};
```

Constants

`gestaltQuickTimeVersion`
The selector you pass to the `Gestalt` function to determine the QuickTime version.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltQuickTime`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

QuickTime Conferencing Information Selector

Specifies information about QuickTime conferencing.

```
enum {
    gestaltQuickTimeConferencingInfo = 'qtci'
};
```

Constants

`gestaltQuickTimeConferencingInfo`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

QuickTime Conferencing Selector

Specifies availability information for QuickTime conferencing.

```
enum {
    gestaltQuickTimeConferencing = 'mtlk'
};
```

QuickTime Streaming Attribute Selector

Specify feature availability information for QuickTime streaming.

```
enum {
    gestaltQuickTimeStreamingFeatures = 'qtsf'
};
```

QuickTime Streaming Version Selector

Specifies version information for QuickTime streaming.

```
enum {
    gestaltQuickTimeStreamingVersion = 'qtst'
};
```

RBV Address Selector

Specifies information about the RBV base address.

```
enum {
    gestaltRBVAddr = 'rbv '
};
```

Realtime Manager Attribute Selectors

Specify feature availability information for the Realtime Manager.

```
enum {
    gestaltRealtimeMgrAttr = 'rtmr',
    gestaltRealtimeMgrPresent = 0
};
```

Constants

`gestaltRealtimeMgrAttr`
The selector you pass to the `Gestalt` function to determine the Realtime Manager attributes.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltRealtimeMgrPresent`
(description forthcoming)
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

Resource Manager Bug Fixes Attribute Selectors

Specify feature availability information for Resource Manager bug fixes.

```
enum {
    gestaltResourceMgrBugFixesAttrs = 'rmbg',
    gestaltRMForceSysHeapRolledIn = 0,
    gestaltRMFakeAppleMenuItemsRolledIn = 1,
    gestaltSanityCheckResourceFiles = 2,
    gestaltSupportsFSpResourceFileAlreadyOpenBit = 3,
    gestaltRMSupportsFSCalls = 4,
    gestaltRMTypeIndexOrderingReverse = 8
};
```

Resource Manager Attribute Selectors

Specify feature availability information for the Resource Manager.

```
enum {
    gestaltResourceMgrAttr = 'rsrc',
    gestaltPartialRsrcs = 0,
    gestaltHasResourceOverrides = 1
};
```

Constants

`gestaltResourceMgrAttr`
The `Gestalt` selector you pass to determine which Resource Manager attributes are present.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltPartialRsrcs`
If `true`, partial resources exist.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltHasResourceOverrides`

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion
Before calling any function dependent upon the Resource Manager, your application should pass the selector `gestaltResourceMgrAttr` to the `Gestalt` function to determine the Resource Manager attributes that are present.

ROM Size Selector

Specifies information about ROM size information.

```
enum {
    gestaltROMSize = 'rom '
};
```

Constants
`gestaltROMSize`

The selector you pass to the `Gestalt` function to determine the size of the installed ROM, in bytes. The value is returned in only one word.

You should not infer the existence of certain hardware or software features from the responses that Gestalt returns when you pass it this selector.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

ROM Version Selector

Specifies ROM version information.

```
enum {
    gestaltROMVersion = 'romv'
};
```

Constants
`gestaltROMVersion`

This selector is NOT supported in Carbon.

The selector you pass to the `Gestalt` function to determine the version number of the installed ROM (in the low-order word of the return value).

Never infer the existence of certain hardware or software features from the responses that `Gestalt` returns when you pass it this selector.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

SCC Read Address Selector

Specifies information about the base address for reading SCC.

```
enum {
    gestaltSCCReadAddr = 'sccr'
};
```

SCC Write Address Selector

Specifies information about the base address for writing SCC.

```
enum {
    gestaltSCCWriteAddr = 'sccw'
};
```

SCSI Manager Attribute Selectors

Specify feature availability information for the SCSI Manager.

```
enum {
    gestaltSCSI = 'scsi',
    gestaltAsyncSCSI = 0,
    gestaltAsyncSCSIINROM = 1,
    gestaltSCSISlotBoot = 2,
    gestaltSCSIPollSIH = 3
};
```

Scrap Manager Selectors

Specify version and feature availability information for the Scrap Manager.

```
enum {
    gestaltScrapMgrAttr = 'scra',
    gestaltScrapMgrTranslationAware = 0
};
```

Constants

`gestaltScrapMgrAttr`
The `Gestalt` selector you pass to determine which Scrap Manager attributes are present.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltScrapMgrTranslationAware`
If `true`, the Scrap Manager supports Translation Manager.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

Discussion

Before calling any function dependent upon the Scrap Manager, your application should pass the selector `gestaltScrapMgrAttr` to the `Gestalt` function to determine the Scrap Manager attributes that are present.

Screen Capture Selectors

Specifies location information for screen capture.

```
enum {
    gestaltScreenCaptureMain = 'pic1',
    gestaltScreenCaptureDir = 'pic2'
};
```

Script Manager Version Selector

Specifies version information for the Script Manager.

```
enum {
    gestaltScriptMgrVersion = 'scri'
};
```

Constants

`gestaltScriptMgrVersion`

The selector you pass to the `Gestalt` function to determine the version number of the Script Manager (in the low-order word of the return value).

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Script Systems Count Selector

Specifies information about the number of active script systems.

```
enum {
    gestaltScriptCount = 'scr#'
};
```

Constants

`gestaltScriptCount`

The selector you pass to the Gestalt function to determine the number of script systems currently active.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Serial Hardware Attribute Selectors

Specify serial hardware attributes.

```
enum {
    gestaltSerialAttr = 'ser ',
    gestaltHasGPIaToDCDa = 0,
    gestaltHasGPIaToRTxCa = 1,
    gestaltHasGPIbToDCDb = 2,
    gestaltHidePortA = 3,
    gestaltHidePortB = 4,
    gestaltPortADisabled = 5,
    gestaltPortBDisabled = 6
};
```

Constants

`gestaltSerialAttr`
The selector you pass to the `Gestalt` function to determine the serial hardware attributes of the machine, such as whether or not the GPIa line is connected and can be used for external clocking.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasGPIaToDCDa`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasGPIaToRTxCa`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasGPIbToDCDb`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHidePortA`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHidePortB`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPortADisabled`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPortBDisabled`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Serial Port Arbitrator Attribute Selectors

Specify feature availability information for serial port arbitration.

```
enum {
    gestaltArbitorAttr = 'arb ',
    gestaltSerialArbitrationExists = 0
};
```

Settings Manager Attribute Selectors

Specify feature availability information for the Settings Manager.

```
enum {
    gestaltALMAttr = 'trip',
    gestaltALMPresent = 0,
    gestaltALMHasSFGroup = 1,
    gestaltALMHasCFMSupport = 2,
    gestaltALMHasRescanNotifiers = 3
};
```

Discussion
See also "Settings Manager Version Selector" (page 1091).

Settings Manager Location Selector

Specifies location information for the Settings Manager.

```
enum {
    gestaltALMHasSFLocation = gestaltALMHasSFGroup
};
```

Settings Manager Version Selector

Specifies version information for the Settings Manager.

```
enum {
    gestaltALMVers = 'walk'
};
```

Shutdown Attribute Selectors

Specify shutdown attributes.

```
enum {
    gestaltShutdownAttributes = 'shut',
    gestaltShutdownHassdOnBootVolUnmount = 0
};
```

Single Window Mode Selectors

Specify single-window modes.

```
enum {
    gestaltHasSingleWindowModeBit = 8,
    gestaltHasSingleWindowModeMask = (1L << gestaltHasSingleWindowModeBit)
};
```

Slot Attribute Selectors

Specify feature availablity for slots.

```
enum {
    gestaltSlotAttr = 'slot',
    gestaltSlotMgrExists = 0,
    gestaltNuBusPresent = 1,
    gestaltSESlotPresent = 2,
    gestaltSE30SlotPresent = 3,
    gestaltPortableSlotPresent = 4
};
```

Constants

`gestaltSlotAttr`
The selector you pass to the `Gestalt` function to determine the Slot Manager attributes.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltSlotMgrExists`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltNuBusPresent`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltSESlotPresent`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltSE30SlotPresent`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltPortableSlotPresent`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

Slot Number Selector

Specifies information about the first physical slot in the computer.

```
enum {
    gestaltFirstSlotNumber = 'slt1'
};
```

Constants

`gestaltFirstSlotNumber`
The first physical slot.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Software Vendor Codes

Specify codes for software vendors.

```
enum {
    gestaltSoftwareVendorCode = 'srad',
    gestaltSoftwareVendorApple = 'Appl',
    gestaltSoftwareVendorLicensee = 'Lcns'
};
```

Sound Manager Attribute Selectors

Specify feature availability information for the Sound Manager.

```
enum {
    gestaltSoundAttr = 'snd ',
    gestaltStereoCapability = 0,
    gestaltStereoMixing = 1,
    gestaltSoundIOMgrPresent = 3,
    gestaltBuiltInSoundInput = 4,
    gestaltHasSoundInputDevice = 5,
    gestaltPlayAndRecord = 6,
    gestalt16BitSoundIO = 7,
    gestaltStereoInput = 8,
    gestaltLineLevelInput = 9,
    gestaltSndPlayDoubleBuffer = 10,
    gestaltMultiChannels = 11,
    gestalt16BitAudioSupport = 12
};
```

Constants

`gestaltSoundAttr`
The Gestalt selector which you pass to the `Gestalt` function.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltStereoCapability`
Set if the built-in sound hardware is able to produce stereo sounds.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltStereoMixing`
Set if the built-in sound hardware mixes both left and right channels of stereo sound into a single audio signal for the internal speaker.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltSoundIOMgrPresent`
Set if the Sound Input Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltBuiltInSoundInput`
Set if a built-in sound input device is available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasSoundInputDevice`
Set if a sound input device is available. This device can be either built-in or external.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPlayAndRecord`
Set if the built-in sound hardware is able to play and record sounds simultaneously. If this bit is clear, the built-in sound hardware can either play or record, but not do both at once. This bit is valid only if the `gestaltBuiltInSoundInput` bit is set, and it applies only to any built-in sound input and output hardware.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestalt16BitSoundIO`
Set if the built-in sound hardware is able to play and record 16-bit samples. This indicates that built-in hardware necessary to handle 16-bit data is available.

This bit is not defined for versions of the Sound Manager prior to version 3.0.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltStereoInput`
Set if the built-in sound hardware can record stereo sounds.

This bit is not defined for versions of the Sound Manager prior to version 3.0.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltLineLevelInput`
Set if the built-in sound input port requires line level input.

This bit is not defined for versions of the Sound Manager prior to version 3.0.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltSndPlayDoubleBuffer`
Set if the Sound Manager supports the play-from-disk functions.

This bit is not defined for versions of the Sound Manager prior to version 3.0.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltMultiChannels`
Set if the Sound Manager supports multiple channels of sound.

This bit is not defined for versions of the Sound Manager prior to version 3.0.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestalt16BitAudioSupport`
Set if the Sound Manager can handle 16-bit audio data. This indicates that software necessary to handle 16-bit data is available.

This bit is not defined for versions of the Sound Manager prior to version 3.0.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion
You can pass the `gestaltSoundAttr` selector to the `Gestalt` function to determine information about the sound input capabilities of a Macintosh computer.

The `Gestalt` function returns information by setting or clearing bits in the `response` parameter. The bits relevant to the Sound Input Manager are defined by constants.

Speech Manager Attribute Selectors

Specify feature availability information for the Speech Manager.

```
enum {
    gestaltSpeechAttr = 'ttsc',
    gestaltSpeechMgrPresent = 0,
    gestaltSpeechHasPPCGlue = 1
};
```

Constants
`gestaltSpeechAttr`
The selector you pass to the `Gestalt` function to determine the Speech Manager attributes.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltSpeechMgrPresent`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltSpeechHasPPCGlue`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Speech Recognition Version Selector

Specifies version information for the Speech Recognition Manager.

```
enum {
    gestaltSpeechRecognitionVersion = 'srtb'
};
```

Speech Recognition Manager Attribute Selectors

Specify feature availability information for the Speech Recognition Manager.

```
enum {
    gestaltSpeechRecognitionAttr = 'srta',
    gestaltDesktopSpeechRecognition = 1,
    gestaltTelephoneSpeechRecognition = 2
};
```

Constants

`gestaltSpeechRecognitionAttr`
The selector which you pass to the `Gestalt` function to determine the Speech Recognition Manager attributes.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltDesktopSpeechRecognition`
If this bit is set, the Speech Recognition Manager supports the desktop microphone.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltTelephoneSpeechRecognition`
If this bit is set, the Speech Recognition Manager supports telephone input. In versions 1.5 and earlier, this bit is always 0.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion

You can pass the `gestaltSpeechRecognitionAttr` selector to the `Gestalt` function to get the attributes of the Speech Recognition Manager. `Gestalt` returns information to you by returning a long word in the `response` parameter. The returned values are defined by these constants.

Standard Directory Find Panel Selector

Specifies version information for the standard directory find panel.

```
enum {
    gestaltSDPFindVersion = 'dfnd'
};
```

Standard Directory Prompt Panel Selector

Specifies version information for the standard directory prompt panel.

```
enum {
    gestaltSDPPromptVersion = 'prpv'
};
```

Standard Directory Version Selector

Specifies version information for the standard directory.

```
enum {
    gestaltSDPStandardDirectoryVersion = 'sdvr'
};
```

Startup Disk Attribute Selectors

Specify feature availability information for the startup disk.

```
enum {
    gestaltSplitOSAttr = 'spos',
    gestaltSplitOSBootDriveIsNetworkVolume = 0,
    gestaltSplitOSAware = 1,
    gestaltSplitOSEnablerVolumeIsDifferentFromBootVolume = 2,
    gestaltSplitOSMachineNameSetToNetworkNameTemp = 3,
    gestaltSplitOSMachineNameStartupDiskIsNonPersistent = 5
};
```

Standard File Attribute Selectors

Specify feature availabity information for Standard File.

```
enum {
    gestaltStandardFileAttr = 'stdf',
    gestaltStandardFile58 = 0,
    gestaltStandardFileTranslationAware = 1,
    gestaltStandardFileHasColorIcons = 2,
    gestaltStandardFileUseGenericIcons = 3,
    gestaltStandardFileHasDynamicVolumeAllocation = 4
};
```

Constants

`gestaltStandardFileAttr`

The selector you pass to the Gestalt function to determine the Standard File Package attributes.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltStandardFile58`

If the `gestaltStandardFile58` flag bit is set, you can call the four new procedures—`StandardPutFile`, `StandardGetFile`, `CustomPutFile`, and `CustomGetFile`—introduced with System 7. (The name of the constant reflects the enabling of selectors 5 through 8 on the trap macro that handles the Standard File Package.)

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltStandardFileTranslationAware`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltStandardFileHasColorIcons`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltStandardFileUseGenericIcons`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltStandardFileHasDynamicVolumeAllocation`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

System Architecture Selectors

Specify the native system architecture.

```
enum {
    gestaltSysArchitecture = 'sysa',
    gestalt68k = 1,
    gestaltPowerPC = 2,
    gestaltIntel = 10
};
```

Constants

`gestaltSysArchitecture`
The selector you pass to the `Gestalt` function to determine the native system architecture.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestalt68k`
If the `Gestalt` function returns `gestalt68k`, the system is a MC680x0 Macintosh.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltPowerPC`
If the `Gestalt` function returns `gestaltPowerPC`, the system is a PowerPC Macintosh.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltIntel`
If the `Gestalt` function returns `gestaltIntel`, the system is is an Intel-based Macintosh.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

System Update Version Selector

Specifies version information for system updates.

```
enum {
    gestaltSystemUpdateVersion = 'sysu'
};
```

System Version Selectors

Specifies version information for the operating system.

```
enum {
    gestaltSystemVersion = 'sysv'
    gestaltSystemVersionMajor = 'sys1',
    gestaltSystemVersionMinor = 'sys2',
    gestaltSystemVersionBugFix = 'sys3'
};
```

Constants

`gestaltSystemVersion`

The selector you pass to the `Gestalt` function to determine the version number of the currently active System file. For systems prior to Mac OS X, the version is represented as four hexadecimal digits in the low-order word of the return value. For example, if your application is running in version 7.0.1, then `Gestalt` returns the value 0x0701. Ignore the high-order word of the returned value. For Mac OS X versions, the representation is as shown in Table 18-1.

Table 18-1 The representation of Mac OS X versions by the Gestalt Manager

Mac OS X Version	Representation
10.0	0x1000
10.1	0x1010
10.2	0x1020
10.3	0x1030
10.4	0x1040

If the values of the minor or bug fix revision are larger than 9, then `gestaltSystemVersion` will substitute the value 9 for them. For example, Mac OS X 10.3.15 will be returned as 0x1039, and Mac OS X 10.10.5 will return 0x1095.

Never infer the existence of certain hardware or software features from the responses that `Gestalt` returns when you pass it this selector.

In Mac OS X v10.4 and later, a better way to get system version information is to use the selectors `gestaltSystemVersionMajor`, `gestaltSystemVersionMinor`, and `gestaltSystemVersionBugFix`, which are listed below. These selectors don't have arbitrary limits on the values returned.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltSystemVersionMajor`
The major system version number. For example, in 10.4.12, this would be the decimal value 10.

Available in Mac OS X v10.3 and later.

Declared in `Gestalt.h`.

`gestaltSystemVersionMinor`
The minor system version number. For example, in 10.4.12, this would be the decimal value 4.

Available in Mac OS X v10.3 and later.

Declared in `Gestalt.h`.

`gestaltSystemVersionBugFix`
The bug fix version number. For example, in 10.4.12, this would be the decimal value 12.

Available in Mac OS X v10.3 and later.

Declared in `Gestalt.h`.

Telephone Manager Attribute Selectors

Specify feature availability information for the Telephone Manager.

```
enum {
    gestaltTeleMgrAttr = 'tele',
    gestaltTeleMgrPresent = 0,
    gestaltTeleMgrPowerPCSupport = 1,
    gestaltTeleMgrSoundStreams = 2,
    gestaltTeleMgrAutoAnswer = 3,
    gestaltTeleMgrIndHandset = 4,
    gestaltTeleMgrSilenceDetect = 5,
    gestaltTeleMgrNewTELNewSupport = 6
};
```

Terminal Manager Attribute Selectors

Specify feature availability information for the Terminal Manager.

```
enum {
    gestaltTermMgrAttr = 'term',
    gestaltTermMgrPresent = 0,
    gestaltTermMgrErrorString = 2
};
```

Constants

`gestaltTermMgrAttr`
The selector you pass to the `Gestalt` function to determine the Terminal Manager attributes.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltTermMgrPresent`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltTermMgrErrorString`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

TextEdit Attribute Selectors

Specify feature availability information for TextEdit.

```
enum {
    gestaltTEAttr = 'teat',
    gestaltTEHasGetHiliteRgn = 0,
    gestaltTESupportsInlineInput = 1,
    gestaltTESupportsTextObjects = 2,
    gestaltTEHasWhiteBackground = 3
};
```

Constants

`gestaltTEAttr`
The `Gestalt` selector you pass to determine whichTextEdit attributes are present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltTEHasGetHiliteRgn`
If true, TextEdit has `TEGetHiliteRgn`.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltTESupportsInlineInput`
If `true`, TextEdit does Inline Input.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltTESupportsTextObjects`
If `true`, TextEdit does Text Objects.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltTEHasWhiteBackground`
If `true`, TextEdit supports overriding the `TERec`' data structure background field to white.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion
Before calling any function dependent upon TextEdit, your application should pass the selector `gestaltTEAttr` to the `Gestalt` function to determine the TextEdit attributes that are present.

TextEdit Version Selectors

Specify version information for TextEdit.

```
enum {
    gestaltTextEditVersion = 'te  ',
    gestaltTE1 = 1,
    gestaltTE2 = 2,
    gestaltTE3 = 3,
    gestaltTE4 = 4,
    gestaltTE5 = 5
};
```

Constants

`gestaltTextEditVersion`
The `Gestalt` selector you pass to determine what version of TextEdit is present.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltTE1`
The version of TextEdit found in Mac IIci ROM.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltTE2`
The version of TextEdit shipped with 6.0.4 Script Systems on Mac IIci (Script bug fixes for Mac IIci).
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltTE3`
The version of TextEdit shipped with 6.0.4 Script Systems (all but Mac IIci).
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltTE4`
The version of TextEdit shipped in System 7.0.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltTE5`
`TextWidthHook` is available in TextEdit.
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

Discussion

To determine the version of the current TextEdit, your application should pass the selector `gestalt` TextEditVersion to the `Gestalt` function.

Text Services Manager Attribute Selectors

Specify feature availability information for the Text Services Manager.

```
enum {
    gestaltTSMgrAttr = 'tsma',
    gestaltTSMDisplayMgrAwareBit = 0,
    gestaltTSMdoesTSMTEBit = 1
};
```

Text Services Manager Version Selectors

Specifies version information for the Text Services Manager.

```
enum {
    gestaltTSMgrVersion = 'tsmv',
    gestaltTSMgr15 = 0x0150,
    gestaltTSMgr20 = 0x0200
};
```

Constants

`gestaltTSMgrVersion`
The selector you pass to the `Gestalt` function to determine the version of the Text Services Manager.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltTSMgr15`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltTSMgr20`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Thread Manager Attribute Selectors

Specify feature availability information for the Thread Manager.

```
enum {
    gestaltThreadMgrAttr = 'thds',
    gestaltThreadMgrPresent = 0,
    gestaltSpecificMatchSupport = 1,
    gestaltThreadsLibraryPresent = 2
};
```

Constants

`gestaltThreadMgrAttr`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltThreadMgrPresent`
This bit is set if the Thread Manager is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltSpecificMatchSupport`
: This bit is set if the Thread Manager supports the allocation of threads based on an exact match with the requested stack size. If this bit is not set, the Thread Manager allocates threads based on the closest match to the requested stack size.

  Available in Mac OS X v10.0 and later.

  Declared in `Gestalt.h`.

`gestaltThreadsLibraryPresent`
: This bit is set if the native version of the threads library has been loaded.

  Available in Mac OS X v10.0 and later.

  Declared in `Gestalt.h`.

Discussion
Before calling any function dependent upon the Thread Manager, your application should pass the selector `gestaltThreadMgrAttr` to the `Gestalt` function to determine the Thread Manager attributes that are present.

Time Manager Version Selectors

Specify version information for the Time Manager.

```
enum {
    gestaltTimeMgrVersion = 'tmgr',
    gestaltStandardTimeMgr = 1,
    gestaltRevisedTimeMgr = 2,
    gestaltExtendedTimeMgr = 3,
    gestaltNativeTimeMgr = 4
};
```

Constants
`gestaltTimeMgrVersion`
: The Gestalt selector you pass to determine what version of the Time Manager is present.

  Available in Mac OS X v10.0 and later.

  Declared in `Gestalt.h`.

`gestaltStandardTimeMgr`
: If this bit is set, the original Time Manager is present.

  Available in Mac OS X v10.0 and later.

  Declared in `Gestalt.h`.

`gestaltRevisedTimeMgr`
: If this bit is set, the revised Time Manager is present.

  Available in Mac OS X v10.0 and later.

  Declared in `Gestalt.h`.

`gestaltExtendedTimeMgr`
: If this bit is set, the extended Time Manager is present.

  Available in Mac OS X v10.0 and later.

  Declared in `Gestalt.h`.

`gestaltNativeTimeMgr`
If this bit is set, the native Time Manager is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion
To determine the version of the current Time Manager, your application should pass the selector `gestaltTimeMgrVersion` to the `Gestalt` function.

Toolbox Trap Table Selector

Specifes base address information for the Toolbox trap dispatch table.

```
enum {
    gestaltToolboxTable = 'tbtt'
};
```

Constants
`gestaltToolboxTable`
The selector you pass to the `Gestalt` function to determine the base address of the Toolbox trap dispatch table.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Toolbox Trap Table (Second Half) Selector

Specifies address information for the second half of the Toolbox trap table.

```
enum {
    gestaltExtToolboxTable = 'xttt'
};
```

Constants
`gestaltExtToolboxTable`
The base address of the second half of the Toolbox trap table if the table is discontiguous. If the table is contiguous, this selector returns 0.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Translation Manager Attribute Selectors

Specify feature availability information for the Translation Manager.

```
enum {
    gestaltTranslationAttr = 'xlat',
    gestaltTranslationMgrExists = 0,
    gestaltTranslationMgrHintOrder = 1,
    gestaltTranslationPPCAvail = 2,
    gestaltTranslationGetPathAPIAvail = 3
};
```

Constants

`gestaltTranslationAttr`
The `Gestalt` selector you pass to determine which Translation Manager attributes are present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltTranslationMgrExists`
If `true`, the Translation Manager is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltTranslationMgrHintOrder`
In earlier versions of the Translation Manager, the scrap hints in the `DoTranslateScrapProcPtr` function were reversed. In later versions, this was fixed. If this bit is true, this bug fix is in effect.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltTranslationPPCAvail`
If `true`, the PowerPC Translation Library is available, and you can call the Translation Manager from native PowerPC code.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltTranslationGetPathAPIAvail`
If true, the functions `GetFileTranslationPath` and `GetPathTranslationDialog` are available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion

Before calling any function dependent upon the Translation Manager your application should pass the selector `gestaltTranslationAttr` to the `Gestalt` function to determine the Translation Manager attributes that are present.

TSME Version Selector

Specifies version information for the Text Services Manager integrated with TextEdit.

```
enum {
    gestaltTE6 = 6
};
```

TSMTE Attribute Selectors

Specify feature availability information for TSMTE.

```
enum {
    gestaltTSMTEAttr = 'tmTE',
    gestaltTSMTEPresent = 0,
    gestaltTSMTE = 0
};
```

TSMTE Version Selectors

Specify version information for TSMTE.

```
enum {
    gestaltTSMTEVersion = 'tmTV',
    gestaltTSMTE1 = 0x0100,
    gestaltTSMTE15 = 0x0150,
    gestaltTSMTE152 = 0x0152
};
```

TV Tuner Attribute Selectors

Specifies feature availabilty information for the TV tuner.

```
enum {
    gestaltTVAttr = 'tv  ',
    gestaltHasTVTuner = 0,
    gestaltHasSoundFader = 1,
    gestaltHasHWClosedCaptioning = 2,
    gestaltHasIRRemote = 3,
    gestaltHasVidDecoderScaler = 4,
    gestaltHasStereoDecoder = 5,
    gestaltHasSerialFader = 6,
    gestaltHasFMTuner = 7,
    gestaltHasSystemIRFunction = 8,
    gestaltIRDisabled = 9,
    gestaltINeedIRPowerOffConfirm = 10,
    gestaltHasZoomedVideo = 11
};
```

UDF Selector

Specifes support information for communication between implementations of UDF .

```
enum {
    gestaltUDFSupport = 'kudf'
};
```

USB Attribute Selectors

Specifies feature availability information for USB.

```
enum {
    gestaltUSBAttr = 'usb ',
    gestaltUSBPresent = 0,
    gestaltUSBHasIsoch = 1
};
```

USB Printer Sharing Version Selectors

Specify version information for USB printer sharing.

```
enum {
    gestaltUSBPrinterSharingVersion = 'zak ',
    gestaltUSBPrinterSharingVersionMask = 0x0000FFFF,
    gestaltUSBPrinterSharingAttr = 'zak ',
    gestaltUSBPrinterSharingAttrMask = 0xFFFF0000,
    gestaltUSBPrinterSharingAttrRunning = 0x80000000,
    gestaltUSBPrinterSharingAttrBooted = 0x40000000
};
```

USB Version Selector

Specifies version information for USB.

```
enum {
    gestaltUSBVersion = 'usbv'
};
```

VIA1 Base Address Selector

Specifies base address information for VIA 1.

```
enum {
    gestaltVIA1Addr = 'via1'
};
```

VIA2 Base Address Selector

Specifies base address information for VIA 2.

```
enum {
    gestaltVIA2Addr = 'via2'
};
```

Virtual Memory Manager Attribute Selectors

Specify feature availability information for the Virtual Memory Manager.

```
enum {
    gestaltVMAttr = 'vm  ',
    gestaltVMPresent = 0,
    gestaltVMHasLockMemoryForOutput = 1,
    gestaltVMFilemappingOn = 3,
    gestaltVMHasPagingControl = 4
};
```

Constants

`gestaltVMAttr`
The `Gestalt` selector you pass to determine the virtual memory attributes that are present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltVMPresent`
If `true`, virtual memory is present.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltVMHasLockMemoryForOutput`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltVMFilemappingOn`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltVMHasPagingControl`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

Discussion

Before calling any function dependent on memory, your application should pass the selector `gestaltVMAttr` to the `Gestalt` function to determine the virtual memory attributes that are present.

Virtual Memory Backing Store Selector

Specifes file reference number information for the VM backing store.

```
enum {
    gestaltVMBackingStoreFileRefNum = 'vmbs'
};
```

Virtual Memory Information Type Selectors

Specifes information about the VM type.

```
enum {
    gestaltVMInfoType = 'vmin',
    gestaltVMInfoSizeStorageType = 0,
    gestaltVMInfoSizeType = 1,
    gestaltVMInfoSimpleType = 2,
    gestaltVMInfoNoneType = 3
};
```

Win32 Attribute Selectors

Specify feature availability information for Win32.

```
enum {
    gestaltX86Features = 'x86f',
    gestaltX86HasFPU = 0,
    gestaltX86HasVME = 1,
    gestaltX86HasDE = 2,
    gestaltX86HasPSE = 3,
    gestaltX86HasTSC = 4,
    gestaltX86HasMSR = 5,
    gestaltX86HasPAE = 6,
    gestaltX86HasMCE = 7,
    gestaltX86HasCX8 = 8,
    gestaltX86HasAPIC = 9,
    gestaltX86Reserved10 = 10,
    gestaltX86HasSEP = 11,
    gestaltX86HasMTRR = 12,
    gestaltX86HasPGE = 13,
    gestaltX86HasMCA = 14,
    gestaltX86HasCMOV = 15,
    gestaltX86HasPAT = 16,
    gestaltX86HasPSE36 = 17,
    gestaltX86HasMMX = 23,
    gestaltX86HasFXSR = 24
};
```

Window Manager Attribute Selectors

Specify feature availability information for the Window Manager.

```
enum {
    gestaltWindowMgrAttr = 'wind',
    gestaltWindowMgrPresent = (1L << 0),
    gestaltWindowMgrPresentBit = 0,
    gestaltExtendedWindowAttributes = 1,
    gestaltExtendedWindowAttributesBit = 1,
    gestaltHasFloatingWindows = 2,
    gestaltHasFloatingWindowsBit = 2,
    gestaltHasWindowBuffering = 3,
    gestaltHasWindowBufferingBit = 3,
    gestaltWindowLiveResizeBit = 4,
    gestaltWindowMinimizeToDockBit = 5,
    gestaltHasWindowShadowsBit = 6,
    gestaltSheetsAreWindowModalBit = 7,
    gestaltFrontWindowMayBeHiddenBit = 8,
    gestaltWindowMgrPresentMask = (1L << gestaltWindowMgrPresentBit),
   gestaltExtendedWindowAttributesMask = (1L << gestaltExtendedWindowAttributesBit),
    gestaltHasFloatingWindowsMask = (1L << gestaltHasFloatingWindowsBit),
    gestaltHasWindowBufferingMask = (1L << gestaltHasWindowBufferingBit),
    gestaltWindowLiveResizeMask = (1L << gestaltWindowLiveResizeBit),
    gestaltWindowMinimizeToDockMask = (1L << gestaltWindowMinimizeToDockBit),
    gestaltHasWindowShadowsMask = (1L << gestaltHasWindowShadowsBit),
    gestaltSheetsAreWindowModalMask = (1L << gestaltSheetsAreWindowModalBit),
    gestaltFrontWindowMayBeHiddenMask = (1L << gestaltFrontWindowMayBeHiddenBit)
};
```

Constants

`gestaltWindowMgrAttr`
The `Gestalt` selector passed to determine what features of the Window Manager are present. This selector is available with Mac OS 8.5 and later. The `Gestalt` function produces a 32-bit value whose bits you should test to determine which Window Manager features are available.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltWindowMgrPresent`
If the bit specified by this mask is set, the Window Manager functionality for Appearance Manager 1.1 is available. This bit is set for Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltWindowMgrPresentBit`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltExtendedWindowAttributes`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltExtendedWindowAttributesBit`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasFloatingWindows`
Available in Mac OS X v10.0 and later.

Declared in `Gestalt.h`.

`gestaltHasFloatingWindowsBit`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltHasWindowBuffering`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltHasWindowBufferingBit`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltWindowLiveResizeBit`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltWindowMinimizeToDockBit`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltHasWindowShadowsBit`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltSheetsAreWindowModalBit`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltFrontWindowMayBeHiddenBit`
Available in Mac OS X v10.2 and later.
Declared in `Gestalt.h`.

`gestaltWindowMgrPresentMask`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltExtendedWindowAttributesMask`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltHasFloatingWindowsMask`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltHasWindowBufferingMask`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltWindowLiveResizeMask`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltWindowMinimizeToDockMask`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltHasWindowShadowsMask`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltSheetsAreWindowModalMask`
Available in Mac OS X v10.0 and later.
Declared in `Gestalt.h`.

`gestaltFrontWindowMayBeHiddenMask`
Available in Mac OS X v10.2 and later.
Declared in `Gestalt.h`.

Discussion
Before calling any functions dependent on the Window Manager, your application should pass the selector `gestaltWindowMgrAttr` to the `Gestalt` function to determine which Window Manager functions are available.

WorldScriptII Version Selectors

Specify version information for WorldScript II.

```
enum {
    gestaltWorldScriptIIVersion = 'doub',
    gestaltWorldScriptIIAttr = 'wsat',
    gestaltWSIICanPrintWithoutPrGeneralBit = 0
};
```

Result Codes

The most common result codes returned by the Gestalt Manager are listed below.

Result Code	Value	Description
`gestaltUnknownErr`	-5550	Specifies an unknown error. Available in Mac OS X v10.0 and later.
`gestaltUndefSelectorErr`	-5551	Specifies an undefined selector was passed to the Gestalt Manager. Available in Mac OS X v10.0 and later.
`gestaltDupSelectorErr`	-5552	Specifies you tried to add an entry that already existed. Available in Mac OS X v10.0 and later.
`gestaltLocationErr`	-5553	Specifies the gestalt function ptr was not in the system heap. Available in Mac OS X v10.0 and later.

Keychain Manager Reference

Framework:	CoreServices/CoreServices.h, Carbon/Carbon.h
Declared in	KeychainCore.h KeychainHI.h

Overview

The Keychain Manager is an API that provides a uniform way for your application to handle passwords for multiple users, multiple databases, or any situation in which a user must enter single or multiple passwords. You can use the Keychain Manager to provide secure storage for a user's passwords, cryptographic keys, and digital certificates.

This document, which describes KeychainLib 2.0, is relevant to you if your application needs to create and manage passwords and other secure data.

Important: Keychain Manager is being phased out and replaced by Keychain Services. Any new development should use Keychain Services. See Keychain Services Reference.

Carbon fully supports the Keychain Manager.

Functions by Task

Getting Information About the Keychain Manager

`KCGetKeychainManagerVersion` (page 1154)
Determines the version of the Keychain Manager installed on the user's system.

Creating and Disposing of Keychain References

`KCMakeKCRefFromAlias` (page 1159)
Creates a keychain reference from a keychain alias.

`KCMakeAliasFromKCRef` (page 1158)
Creates an alias to a keychain reference.

`KCReleaseKeychain` (page 1162)
Disposes of the memory associated with a keychain reference.

KCMakeKCRefFromFSSpec (page 1159) Deprecated in Mac OS X v10.5
Creates a keychain reference from a file specification record.

Managing Keychains

KCCreateKeychain (page 1134)
Creates an empty keychain.

kccreatekeychain (page 1135)

KCSetDefaultKeychain (page 1166)
Sets the default keychain.

KCGetDefaultKeychain (page 1152)
Obtains the default keychain.

KCGetStatus (page 1156)
Determines the permissions that are set in a keychain.

KCGetKeychainName (page 1155)
Determines the name of a keychain.

kcgetkeychainname (page 1155)

KCCountKeychains (page 1133)
Determines the number of available keychains.

KCGetIndKeychain (page 1152)
Obtains the reference to an indexed keychain.

Storing and Retrieving Passwords

KCAddAppleSharePassword (page 1120)
Adds a new AppleShare server password to the default keychain.

kcaddapplesharepassword (page 1122)

KCFindAppleSharePassword (page 1136)
Finds the first AppleShare password in the default keychain that matches the specified parameters.

kcfindapplesharepassword (page 1138)

KCAddInternetPassword (page 1126)
Adds a new Internet server password to the default keychain.

kcaddinternetpassword (page 1127)

KCAddInternetPasswordWithPath (page 1128)
Adds a new Internet server password with a specified path to the default keychain.

kcaddinternetpasswordwithpath (page 1129)

`KCFindInternetPassword` (page 1142)
Finds the first Internet password in the default keychain that matches the specified parameters.
`kcfindinternetpassword` (page 1144)

`KCFindInternetPasswordWithPath` (page 1145)
Finds the first Internet password in the default keychain that matches the specified parameters, including path information.
`kcfindinternetpasswordwithpath` (page 1147)

`KCAddGenericPassword` (page 1124)
Adds a new generic password to the default keychain.
`kcaddgenericpassword` (page 1125)

`KCFindGenericPassword` (page 1140)
Finds the first generic password in the default keychain matching the specified parameters.
`kcfindgenericpassword` (page 1142)

Creating and Disposing of Keychain Item References

`KCNewItem` (page 1160)
Creates a reference to a keychain item.
`KCReleaseItem` (page 1161)
Disposes of the memory occupied by a keychain item reference.

Manipulating Keychain Items

`KCAddItem` (page 1130)
Adds a password or other keychain item to the default keychain.
`KCDeleteItem` (page 1136)
Deletes a password or other keychain item from the default keychain.
`KCUpdateItem` (page 1169)
Updates a password or other keychain item.
`KCCopyItem` (page 1132)
Copies a password or other keychain item from one keychain to another.
`KCGetKeychain` (page 1153)
Determines the location of a password or other keychain item.

Setting and Obtaining Keychain Item Data

`KCSetAttribute` (page 1164)
Sets or edits keychain item data using a keychain item attribute structure.

`KCGetAttribute` (page 1149)
Determines keychain item data using a keychain item attribute structure.

`KCSetData` (page 1165)
Sets or edits keychain item data.

`KCGetData` (page 1151)
Determines keychain item data.

Searching for Keychain Items

`KCFindFirstItem` (page 1139)
Finds the first keychain item in a specified keychain that matches specified attributes.

`KCFindNextItem` (page 1148)
Finds the next keychain item matching the previously specified search criteria.

`KCReleaseSearch` (page 1162)
Disposes of the memory occupied by a search criteria reference.

Managing User Interaction

`KCLock` (page 1157)
Locks a keychain.

`KCUnlock` (page 1167)
Displays a dialog box that prompts the user for a password before unlocking a keychain.

`kcunlock` (page 1169)

`KCChangeSettings` (page 1131)
Displays a dialog box enabling the user to change the name, password, or settings of a keychain.

`KCSetInteractionAllowed` (page 1167)
Enables or disables Keychain Manager functions that display a user interface.

`KCIsInteractionAllowed` (page 1157)
Indicates whether Keychain Manager functions that display a user interaction will do so.

Registering Your Keychain Event Callback Function

`KCAddCallback` (page 1123)
Registers your keychain event callback function.

`KCRemoveCallback` (page 1163)
Unregisters your keychain event callback function.

Working With Your Keychain Manager Callback Function

`NewKCCallbackUPP` (page 1170)
Creates a UPP to your keychain event callback.

`InvokeKCCallbackUPP` (page 1119)
Invokes your keychain event callback.

`DisposeKCCallbackUPP` (page 1119)
Disposes of a UPP to your keychain event callback.

Unsupported Functions

`KCChooseCertificate` (page 1132)
Displays a list of certificates that the user can choose from.

`KCFindX509Certificates` (page 1149)
Finds the certificates in a keychain that match specified search criteria.

Functions

DisposeKCCallbackUPP

Disposes of a UPP to your keychain event callback.

Not recommended

```
void DisposeKCCallbackUPP (
   KCCallbackUPP userUPP
);
```

Parameters

userUPP
A Universal Procedure Pointer (UPP) to your keychain event callback function.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194).

Discussion
When you are finished with a UPP to your keychain event callback function, you should dispose of it by calling the `DisposeKCCallbackUPP` function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

There is no replacement function available.

Declared In
`KeychainCore.h`

InvokeKCCallbackUPP

Invokes your keychain event callback.

Not recommended

```
OSStatus InvokeKCCallbackUPP (
   KCEvent keychainEvent,
   KCCallbackInfo *info,
   void *userContext,
   KCCallbackUPP userUPP
);
```

Parameters

keychainEvent

The keychain events you want your application to receive. See "Keychain Events Constants" (page 1180) for a description of possible values. The Keychain Manager tests the bitmask you pass in the `eventMask` parameter of the function `KCAddCallback` (page 1123) to determine which events to pass to your callback function. See "Keychain Events Mask" (page 1182) for a description of this bitmask.

info

A pointer to a structure of type `KCCallbackInfo` (page 1173) that provides information about the keychain event to your callback function. The Keychain Manager passes a pointer to this structure in the `info` parameter of your callback function.

userContext

A pointer to application-defined storage. The Keychain Manager passes this value in the `userContext` parameter of your callback function. Your application can use this to associate any particular call of the `InvokeKCCallbackUPP` function with any particular call of the keychain event callback function.

userUPP

A Universal Procedure Pointer to your keychain event callback function. For information on how to create a keychain event callback function, see `KCCallbackProcPtr` (page 1171).

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194).

Discussion

You should not need to use the function `InvokeKCCallbackUPP`, as the system calls your `KCAddCallback` (page 1123) callback function for you.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

There is no replacement function available.

Declared In

`KeychainCore.h`

KCAddAppleSharePassword

Adds a new AppleShare server password to the default keychain.

Not recommended

```
OSStatus KCAddAppleSharePassword (
   AFPServerSignature *serverSignature,
   StringPtr serverAddress,
   StringPtr serverName,
   StringPtr volumeName,
   StringPtr accountName,
   UInt32 passwordLength,
   const void *passwordData,
   KCItemRef *item
);
```

Parameters

serverSignature

A pointer to a 16-byte Apple File Protocol server signature block. Pass a value of type `AFPServerSignature` (page 1172). Pass `NULL` to match any server signature. The Keychain Manager identifies the location for the password by the information passed in the `serverAddress` and `serverSignature` parameters. You must pass a valid value in at least one of these parameters.

serverAddress

A pointer to a Pascal string containing the server address, which may be specified as an AppleTalk zone name, a DNS domain name (in the format "xxx.yyy.zzz"), or an IP address (in the format "111.222.333.444"). The Keychain Manager identifies the location for the password by the information passed in the `serverAddress` and `serverSignature` parameters. You must pass a valid value in at least one of these parameters.

serverName

A pointer to a Pascal string containing the server name.

volumeName

A pointer to a Pascal string containing the volume name.

accountName

A pointer to a Pascal string containing the account name.

passwordLength

The length of the buffer pointed to by `passwordData`.

passwordData

A pointer to a buffer which will hold the returned password data. Before calling `KCAddAppleSharePassword`, allocate enough memory for the buffer to hold the data you want to store.

item

On return, a pointer to a reference to the added item. Pass `NULL` if you don't want to obtain this reference.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCDuplicateItem` indicates that you tried to add a password that already exists in the keychain. The result code `errKCDataTooLarge` indicates that you tried to add more data than is allowed for a record of this type.

Discussion

The `KCAddAppleSharePassword` function adds a new AppleShare server password to the default keychain that is uniquely identified by the `serverName`, `volumeName`, and `accountName` parameters, and a location specified either by the `serverAddress` or `serverSignature` parameters. The `KCAddAppleSharePassword` function optionally returns a reference to the newly added item.

Most applications do not need to store AppleShare password data, as this is handled transparently by the AppleShare client software. To be compatible with the AppleShare client, you should store a fully-specified File Manager structure `AFPXVolMountInfo` as the password data.

The `KCAddAppleSharePassword` function automatically calls the function `KCUnlock` (page 1167) to display the Unlock Keychain dialog box if the keychain containing the password is currently locked.

You can also call the function `kcaddapplesharepassword` to add an AppleShare server password to the default keychain. `kcaddapplesharepassword` requires that you pass a pointer to a C string instead of a pointer to a Pascal string for the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes
Available beginning with KeychainLib 1.0. In KeychainLib 1.0, the `kcaddapplesharepassword` function provides the same functionality as `KCAddAppleSharePassword`, except that it accepts C strings rather than Pascal strings as arguments. In KeychainLib 2.0, you should use `KCAddAppleSharePassword`, since `kcaddapplesharepassword` is provided for convenience only and may be removed from the header file at some point in the future.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

It is recommended that you use internet passwords instead of AppleShare passwords. Use the `SecKeychainAddInternetPassword` function in Keychain Services instead.

Declared In
`KeychainHI.h`

kcaddapplesharepassword

Not recommended

```
OSStatus kcaddapplesharepassword (
   AFPServerSignature *serverSignature,
   const char *serverAddress,
   const char *serverName,
   const char *volumeName,
   const char *accountName,
   UInt32 passwordLength,
   const void *passwordData,
   KCItemRef *item
);
```

Discussion
This function is available for convenience only and may be removed. Use the function `KCAddAppleSharePassword` (page 1120) instead.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

It is recommended that you use internet passwords instead of AppleShare passwords. Use the `SecKeychainAddInternetPassword` function in Keychain Services instead.

Declared In
`KeychainHI.h`

KCAddCallback

Registers your keychain event callback function.

Not recommended

```
OSStatus KCAddCallback (
   KCCallbackUPP callbackProc,
   KCEventMask eventMask,
   void *userContext
);
```

Parameters

`callbackProc`

A Universal Procedure Pointer (UPP) to your keychain event callback function, described in `KCCallbackProcPtr` (page 1171). You indicate the type of keychain events you want to receive by passing a bitmask of the desired events in the `eventMask` parameter. To create a UPP to your callback function, call the function `NewKCCallbackUPP` (page 1170).

`eventMask`

A bitmask indicating the keychain events that your application wishes to be notified of. See "Keychain Events Mask" (page 1182) for a description of this bitmask. The Keychain Manager tests this mask to determine the keychain events that you wish to receive, and passes these events in the `keychainEvent` parameter of your callback function. See "Keychain Events Constants" (page 1180) for a description of these events.

`userContext`

A pointer to application-defined storage that will be passed to your callback function. Your application can use this to associate any particular call of the `KCAddCallback` function with any particular call of your keychain event callback function.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCDuplicateCallback` indicates that your callback function is already registered.

Discussion
You can register your callback function by passing a UPP to it in the `callbackProc` parameter of the `KCAddCallback` function. Once you have done so, the Keychain Manager calls the function `InvokeKCCallbackUPP` (page 1119) when the keychain event specified in the `eventMask` parameter occurs. In turn, the function `InvokeKCCallbackUPP` (page 1119) passes the keychain event, information about the event, and application-defined storage to your keychain event callback function.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainAddCallback` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCAddGenericPassword

Adds a new generic password to the default keychain.

Not recommended

```
OSStatus KCAddGenericPassword (
   StringPtr serviceName,
   StringPtr accountName,
   UInt32 passwordLength,
   const void *passwordData,
   KCItemRef *item
);
```

Parameters

serviceName
A pointer to a Pascal string containing an application-defined service name.

accountName
A pointer to a Pascal string containing an application-defined account name.

passwordLength
The length of the password data to be stored

passwordData
A pointer to the buffer that holds the returned password data. Before calling the `KCAddGenericPassword` function, allocate enough memory for the buffer to hold the data you want to store.

item
On return, a pointer to a reference to the added item. Pass `NULL` if you don't want to obtain this reference.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCDuplicateItem` indicates that you tried to add a password that already exists in the keychain. The result code `errKCDataTooLarge` indicates that you tried to add more data than is allowed for a record of this type.

Discussion
The `KCAddGenericPassword` function adds a new generic password to the default keychain. Required parameters to identify the password are `serviceName` and `accountName`, which are application-defined strings. The `KCAddGenericPassword` function optionally returns a reference to the newly added item.

You can use the `KCAddGenericPassword` function to add passwords for accounts other than Internet or AppleShare. For example, you might add passwords for your database or scheduling programs.

You can also call the function `kcaddgenericpassword` to add a new generic password to the default keychain. The difference between the two functions is that the `kcaddgenericpassword` function takes a pointer to a C string instead of a Pascal string in the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

The `KCAddGenericPassword` function automatically calls the function `KCUnlock` (page 1167) to display the Unlock Keychain dialog box if the keychain containing the password is currently locked.

Version Notes
Available beginning with KeychainLib 1.0. In KeychainLib 1.0, the `kcaddgenericpassword` function provides the same functionality as the function `KCAddGenericPassword`, except that it accepts C strings rather than Pascal strings as arguments. In KeychainLib 2.0, you should use the `KCAddGenericPassword` function, since the `kcaddgenericpassword` function is provided for convenience only and may be removed from the header file at some point in the future.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainAddGenericPassword` function in Keychain Services instead.

Declared In
`KeychainHI.h`

kcaddgenericpassword

Not recommended

```
OSStatus kcaddgenericpassword (
   const char *serviceName,
   const char *accountName,
   UInt32 passwordLength,
   const void *passwordData,
   KCItemRef *item
);
```

Discussion
This function is available for convenience only and may be removed. Use the function `KCAddGenericPassword` (page 1124) instead.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainAddGenericPassword` function in Keychain Services instead.

Declared In
`KeychainHI.h`

KCAddInternetPassword

Adds a new Internet server password to the default keychain.

Not recommended

```
OSStatus KCAddInternetPassword (
   StringPtr serverName,
   StringPtr securityDomain,
   StringPtr accountName,
   UInt16 port,
   OSType protocol,
   OSType authType,
   UInt32 passwordLength,
   const void *passwordData,
   KCItemRef *item
);
```

Parameters

serverName

A pointer to a Pascal string containing the server name.

securityDomain

A pointer to a Pascal string containing the security domain. This parameter is optional, as not all protocols will require it.

accountName

A pointer to a Pascal string containing the account name.

port

The TCP/IP port number. Pass the constant `kAnyPort`, described in "Default Internet Port Constant" (page 1180), to specify any port.

protocol

The protocol associated with this password. See "Keychain Protocol Type Constants" (page 1191) for a description of possible values. Pass the constant `kAnyProtocol`, described in "Default Internet Protocol And Authentication Type Constants" (page 1180), to specify any protocol.

authType

The authentication scheme used. See "Authentication Type Constants" (page 1176) for a description of possible values. Pass the constant `kAnyAuthType`, described in "Default Internet Protocol And Authentication Type Constants" (page 1180), to specify any authentication scheme.

passwordLength

The length of the buffer pointed to by `passwordData`.

passwordData

A pointer to a buffer that holds the returned password data. Before calling the `KCAddInternetPasswordWithPath` function, allocate enough memory for the buffer to hold the data you want to store.

item

On return, a pointer to a reference to the added item. Pass `NULL` if you don't want to obtain this reference.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCDuplicateItem` indicates that you tried to add a password that already exists in the keychain. The result code `errKCDataTooLarge` indicates that you tried to add more data than is allowed for a record of this type.

Discussion
The `KCAddInternetPassword` function adds a new Internet server password to the default keychain. Required parameters to identify the password are `serviceName` and `accountName` (you cannot pass `NULL` for both parameters). In addition, some protocols may require an optional value in the `securityDomain` parameter when authentication is requested. `KCAddInternetPassword` optionally returns a reference to the newly added item.

The `KCAddInternetPassword` function automatically calls the function `KCUnlock` (page 1167) to display the Unlock Keychain dialog box if the keychain containing the password is currently locked.

You can also call the function `kcaddinternetpassword` to add a new Internet server password to the default keychain. The `kcaddinternetpassword` function requires that you pass a pointer to a C string instead of a pointer to a Pascal string for the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes
Available beginning with KeychainLib 1.0. In KeychainLib 1.0, the `kcaddinternetpassword` function provides the same functionality as `KCAddInternetPassword`, except that it accepts C strings rather than Pascal strings as arguments. In KeychainLib 2.0, you should use `KCAddInternetPassword`, since `kcaddinternetpassword` is provided for convenience only and may be removed from the header file at some point in the future.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainAddInternetPassword` function in Keychain Services instead.

Declared In
`KeychainHI.h`

kcaddinternetpassword

Not recommended

```
OSStatus kcaddinternetpassword (
   const char *serverName,
   const char *securityDomain,
   const char *accountName,
   UInt16 port,
   OSType protocol,
   OSType authType,
   UInt32 passwordLength,
   const void *passwordData,
   KCItemRef *item
);
```

Discussion
This function is available for convenience only and may be removed. Use the function `KCAddInternetPassword` (page 1126) instead.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainAddInternetPassword` function in Keychain Services instead.

Declared In

`KeychainHI.h`

KCAddInternetPasswordWithPath

Adds a new Internet server password with a specified path to the default keychain.

Not recommended

```
OSStatus KCAddInternetPasswordWithPath (
   StringPtr serverName,
   StringPtr securityDomain,
   StringPtr accountName,
   StringPtr path,
   UInt16 port,
   OSType protocol,
   OSType authType,
   UInt32 passwordLength,
   const void *passwordData,
   KCItemRef *item
);
```

Parameters

serverName

A pointer to a Pascal string containing the server name.

securityDomain

A pointer to a Pascal string containing the security domain. This parameter is optional, as not all protocols will require it.

accountName

A pointer to a Pascal string containing the account name.

path

A pointer to a Pascal string containing additional information that specifies a file or directory on the server specified by the `serverName` parameter. In a typical URL, path information begins directly after the first slash ("/") character following the server name. This parameter is optional.

port

The TCP/IP port number. Pass the constant `kAnyPort`, described in "Default Internet Port Constant" (page 1180), to specify any port.

protocol

The protocol associated with this password. See "Keychain Protocol Type Constants" (page 1191) for a description of possible values. Pass the constant `kAnyProtocol`, described in "Default Internet Protocol And Authentication Type Constants" (page 1180), to specify any protocol.

authType

The authentication scheme used. See "Authentication Type Constants" (page 1176) for a description of possible values. Pass the constant `kAnyAuthType`, described in "Default Internet Protocol And Authentication Type Constants" (page 1180), to specify any authentication scheme.

passwordLength

The length of the buffer pointed to by `passwordData`.

passwordData

A pointer to a buffer which will hold the returned password data. Before calling `KCAddInternetPasswordWithPath`, allocate enough memory for the buffer to hold the data you want to store.

item

On return, a pointer to a reference to the added item. Pass `NULL` if you don't want to obtain this reference.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCDuplicateItem` indicates that you tried to add a password that already exists in the keychain. The result code `errKCDataTooLarge` indicates that you tried to add more data than is allowed for a record of this type.

Discussion

The `KCAddInternetPasswordWithPath` function enables you to specify path information when adding a new Internet server password to the default keychain. Required parameters to identify the password are `serviceName` and `accountName` (you cannot pass `NULL` for both parameters). In addition, some protocols may require an optional `securityDomain` when authentication is requested. `KCAddInternetPasswordWithPath` optionally returns a reference to the newly added item.

The `KCAddInternetPasswordWithPath` function automatically calls the function `KCUnlock` (page 1167) to display the Unlock Keychain dialog box if the keychain containing the password is currently locked.

You can also call the function `kcaddinternetpasswordwithpath` to add a new Internet server password to the default keychain. The function `kcaddinternetpasswordwithpath` requires that you pass a pointer to a C string instead of a pointer to a Pascal string for the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes

Available beginning with KeychainLib 2.0. In KeychainLib 1.0, the `kcaddinternetpasswordwithpath` function provides the same functionality as the `KCAddInternetPasswordWithPath` function, except that it accepts C strings rather than Pascal strings as arguments. In KeychainLib 2.0, you should use the `KCAddInternetPasswordWithPath` function, since the function `kcaddinternetpasswordwithpath` is provided for convenience only and may be removed from the header file at some point in the future.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainAddInternetPassword` function in Keychain Services instead.

Declared In

`KeychainHI.h`

kcaddinternetpasswordwithpath

Not recommended

```
OSStatus kcaddinternetpasswordwithpath (
   const char *serverName,
   const char *securityDomain,
   const char *accountName,
   const char *path,
   UInt16 port,
   OSType protocol,
   OSType authType,
   UInt32 passwordLength,
   const void *passwordData,
   KCItemRef *item
);
```

Discussion
This function is available for convenience only and may be removed. Use the function `KCAddInternetPasswordWithPath` (page 1128) instead.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainAddInternetPassword` function in Keychain Services instead.

Declared In
`KeychainHI.h`

KCAddItem

Adds a password or other keychain item to the default keychain.

Not recommended

```
OSStatus KCAddItem (
   KCItemRef item
);
```

Parameters

item

A reference to the keychain item you wish to add. If you pass an existing item in the keychain, the item is updated. If you pass an item that has not been previously added to the keychain and an identical item already exists in the keychain, `KCAddItem` returns the result code `errKCDuplicateItem`.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid. The result code `errKCDuplicateItem` indicates that you tried to add a new item that already exists in the keychain.

Discussion
You can use the `KCAddItem` function to add a password or other keychain item to the permanent data store of the default keychain. If you want to add a password to a keychain other than the default, call the function `KCSetDefaultKeychain` (page 1166) to change the default keychain. The `KCAddItem` function automatically calls the function `KCUnlock` (page 1167) to display the Unlock Keychain dialog box if the keychain containing the item is currently locked.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

When you use Keychain Services to create an item, it is always added to the specified keychain at creation time.

Declared In
`KeychainHI.h`

KCChangeSettings

Displays a dialog box enabling the user to change the name, password, or settings of a keychain.

Not recommended

```
OSStatus KCChangeSettings (
   KCRef keychain
);
```

Parameters

keychain
A reference to an unlocked keychain. Pass in `NULL` to specify the default keychain.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errUserCanceled` indicates that the user pressed the Cancel button in the Change Settings dialog box. The result code `errKCNoDefaultKeychain` indicates that the default keychain could not be found. The result code `errKCInvalidKeychain` indicates that the specified keychain is invalid.

Discussion
Typically, your application should not call the `KCChangeSettings` function. You would only call the `KCChangeSettings` function in response to a user's request to change keychain settings, name, or password. Note that you cannot change a keychain passphrase directly. You must call the `KCChangeSettings` function and allow the user to change it.

Version Notes
Available beginning with KeychainLib 2.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainSetSettings` function in Keychain Services instead.

Declared In
`KeychainHI.h`

KCChooseCertificate

Displays a list of certificates that the user can choose from.

Unsupported

```
OSStatus KCChooseCertificate (
    CFArrayRef items,
    KCItemRef *certificate,
    CFArrayRef policyOIDs,
    KCVerifyStopOn stopOn
);
```

Parameters

items

An array of certificate references.

certificate

If the `items` array only contains one certificate, on return, a pointer to that certificate. In this case, no user interface is displayed.

policyOIDs

An array of trust policy options used for Macintosh file signing. To obtain a pointer to this array, call the function `SecMacGetDefaultPolicyOIDs`.

stopOn

The criteria to use in selecting the certificates to display. See "Certificate Verification Criteria" (page 1179) for a description of this mask.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). The result code `userCanceledErr` indicates that the user pressed the Cancel button in the user interface.

Discussion

The `KCChooseCertificate` function displays a list of the certificates from which the user can choose. If only one certificate matches the criteria, the reference is passed back in the `certificate` parameter and no user interface is presented.

Version Notes

Available beginning with KeychainLib 2.0.

Carbon Porting Notes

This function is obsolete. There is currently no replacement.

Declared In

`KeychainHI.h`

KCCopyItem

Copies a password or other keychain item from one keychain to another.

Not recommended

```
OSStatus KCCopyItem (
   KCItemRef item,
   KCRef destKeychain,
   KCItemRef *copy
);
```

Parameters

item

A reference to the keychain item you wish to copy.

destKeychain

A reference to the keychain into which the item is to be copied.

copy

A pointer to a reference to the new copied keychain item.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCReadOnly` indicates that the destination keychain is read only. The result code `errKCNoSuchClass` indicates that the item has an invalid keychain item class. The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid.

Discussion

You can use the `KCCopyItem` function to copy a keychain item from one keychain to another. The `KCCopyItem` function automatically calls the function `KCUnlock` (page 1167) to display the Unlock Keychain dialog box if the keychain containing the item to be copied is currently locked.

Version Notes

Available beginning with KeychainLib 2.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemCopyContent` function in Keychain Services instead.

Declared In

`KeychainCore.h`

KCCountKeychains

Determines the number of available keychains.

Not recommended

```
UInt16 KCCountKeychains (
   void
);
```

Parameters

Return Value

The number of available keychains. This includes all keychains in the Keychains folder, as well as any other keychains known to the Keychain Manager.

Discussion
This function reports the number of keychains known to the Keychain Manager. These keychains are created by the function `KCCreateKeychain` (page 1134).

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainCopySearchList` function in Keychain Services followed by a call to `CFArrayGetCount` instead.

Declared In
`KeychainCore.h`

KCCreateKeychain

Creates an empty keychain.

Not recommended

```
OSStatus KCCreateKeychain (
   StringPtr password,
   KCRef *keychain
);
```

Parameters

password

A pointer to a Pascal string representing the password string which will be used to protect the new keychain. If you pass `NULL`, the Keychain Setup dialog box will be displayed to obtain it.

keychain

A pointer to a reference to the keychain you wish to create. You create a keychain reference by calling the function `KCMakeKCRefFromFSSpec` (page 1159) or `KCMakeKCRefFromAlias` (page 1159). If you pass a pointer to a keychain reference, the user will not need to be prompted for a name and location; in all other cases, `KCCreateKeychain` will interactively request this information from the user. If you pass a pointer to a `NULL` keychain reference, the Keychain Manager allocates the memory for the keychain reference and returns it in this parameter. Pass a `NULL` pointer if you do not need a reference returned.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194). The result code `userCanceledErr` indicates that the user pressed the Cancel button in the create keychain. The result code `errKCDuplicateKeychain` indicates that the user tried to create a keychain which already exists. The result code `errKCInvalidKeychain` indicates that the specified keychain is invalid. Additional errors may be returned if the keychain could not be created (for example, a file system or network error may be returned if there is no write access to the storage media).

Discussion
The `KCCreateKeychain` function creates an empty keychain. The `keychain` and `password` parameters are optional. If user interaction to create a keychain is posted, the newly-created keychain is automatically unlocked after creation.

You can also call the function `kccreatekeychain` to create an empty keychain. The function `kccreatekeychain` requires that you pass a pointer to a C string instead of a pointer to a Pascal string in the `password` parameter.

Special Considerations

It is recommended that the `KCCreateKeychain` function not be explicitly called by applications. Instead, you should call one of the add functions in "Storing and Retrieving Passwords" (page 1116) to add a password to the default keychain. If a default keychain does not exist, it is created automatically.

When you are finished with a keychain, you must deallocate its memory by calling the function `KCReleaseKeychain` (page 1162).

Version Notes

Available beginning with KeychainLib 1.0. In KeychainLib 1.0, the `kccreatekeychain` function provides the same functionality as `KCCreateKeychain`. In KeychainLib 2.0, you should use `KCCreateKeychain`, since `kccreatekeychain` is provided for convenience only and may be removed from the header file at some point in the future.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainCreate` function in Keychain Services instead.

Declared In

`KeychainHI.h`

kccreatekeychain

Not recommended

```
OSStatus kccreatekeychain (
   const char *password,
   KCRef *keychain
);
```

Discussion

This function is available for convenience only and may be removed. Use the function `KCCreateKeychain` (page 1134) instead.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainCreate` function in Keychain Services instead.

Declared In

`KeychainHI.h`

KCDeleteItem

Deletes a password or other keychain item from the default keychain.

Not recommended

```
OSStatus KCDeleteItem (
   KCItemRef item
);
```

Parameters

item

A reference to the keychain item you wish to delete. If you pass an item that has not been previously added to the keychain, the function `KCDeleteItem` does nothing and returns `noErr`.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid.

Discussion

You can use the `KCDeleteItem` function to delete a keychain item from the permanent data store of the default keychain. The `KCDeleteItem` function automatically calls the function `KCUnlock` (page 1167) to display the Unlock Keychain dialog box if the keychain containing the item is currently locked.

Special Considerations

The `KCDeleteItem` function does not dispose the memory occupied by the item reference. To do so, call the function `KCReleaseItem` (page 1161) when you are finished with an item.

Version Notes

Available beginning with KeychainLib 1.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemDelete` function in Keychain Services instead.

Declared In

`KeychainCore.h`

KCFindAppleSharePassword

Finds the first AppleShare password in the default keychain that matches the specified parameters.

Not recommended

```
OSStatus KCFindAppleSharePassword (
   AFPServerSignature *serverSignature,
   ConstStringPtr serverAddress,
   ConstStringPtr serverName,
   ConstStringPtr volumeName,
   ConstStringPtr accountName,
   UInt32 maxLength,
   void *passwordData,
   UInt32 *actualLength,
   KCItemRef *item
);
```

Parameters

serverSignature

A pointer to a 16-byte Apple File Protocol server signature block. Pass a value of type `AFPServerSignature` (page 1172). Pass `NULL` to match any server signature. The Keychain Manager identifies the location for the password by the information passed in the `serverAddress` and `serverSignature` parameters. You must pass a valid value in at least one of these parameters.

serverAddress

A pointer to a Pascal string containing the server address, which may be specified as an AppleTalk zone name, a DNS domain name (in the format "xxx.yyy.zzz"), or an IP address (in the format "111.222.333.444"). The Keychain Manager identifies the location for the password by the information passed in the `serverAddress` and `serverSignature` parameters. You must pass a valid value in at least one of these parameters.

serverName

A pointer to a Pascal string containing the server name. Pass `NULL` to match any server name.

volumeName

A pointer to a Pascal string containing the volume name. Pass `NULL` to match any volume name.

accountName

A pointer to a Pascal string containing the account name. Pass `NULL` to match any account name.

maxLength

The length of the buffer pointed to by `passwordData`.

passwordData

A pointer to a buffer which will hold the returned password data. Before calling `KCFindAppleSharePassword`, allocate enough memory for the buffer to hold the data you want to store. Pass `NULL` if you want to obtain the item reference but not the password data. In this case, you must also pass `NULL` in the `actualLength` parameter. On return, a pointer to the returned password data.

actualLength

On return, the actual length of the password data that was retrieved. If the buffer pointed to by `passwordData` is smaller than the actual length of the data, `KCFindAppleSharePassword` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling `KCFindAppleSharePassword` again.

item

On return, a pointer to a reference to the found item. Pass `NULL` if you don't want to obtain this reference.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCNoDefaultKeychain` indicates that no default keychain was found. The result code `errKCItemNotFound` indicates that no matching password item was found. The result code `errKCBufferTooSmall` indicates that your application must allocate a new buffer of sufficient size before calling `KCFindAppleSharePassword` again.

Discussion
The `KCFindAppleSharePassword` function finds the first AppleShare password item which matches the attributes you provide. The buffer specified in the `passwordData` parameter must be large enough to hold the password data, otherwise `KCFindAppleSharePassword` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling the `KCFindAppleSharePassword` function again. The `KCFindAppleSharePassword` function optionally returns a reference to the found item.

The `KCFindAppleSharePassword` function automatically calls the function `KCUnlock` (page 1167) to display the Unlock Keychain dialog box if the keychain containing the password is currently locked.

You can also call the function `kcfindapplesharepassword` to find the first AppleShare server password matching specified attributes. `kcfindapplesharepassword` requires that you pass a pointer to a C string instead of a pointer to a Pascal string for the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes
Available beginning with KeychainLib 1.0. In KeychainLib 1.0, the `kcfindapplesharepassword` function provides the same functionality as `KCFindAppleSharePassword`, except that it accepts C strings rather than Pascal strings as arguments. In KeychainLib 2.0, you should use `KCFindAppleSharePassword`, since `kcfindapplesharepassword` is provided for convenience only and may be removed from the header file at some point in the future.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the Keychain Services function `SecKeychainSearchCreateFromAttributes` followed by the `SecKeychainSearchCopyNext` function instead.

Declared In
`KeychainCore.h`

kcfindapplesharepassword
Not recommended

```
OSStatus kcfindapplesharepassword (
   AFPServerSignature *serverSignature,
   const char *serverAddress,
   const char *serverName,
   const char *volumeName,
   const char *accountName,
   UInt32 maxLength,
   void *passwordData,
   UInt32 *actualLength,
   KCItemRef *item
);
```

Discussion
This function is available for convenience only and may be removed. Use the function `KCFindAppleSharePassword` (page 1136) instead.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the Keychain Services function `SecKeychainSearchCreateFromAttributes` followed by the `SecKeychainSearchCopyNext` function instead.

Declared In
`KeychainCore.h`

KCFindFirstItem

Finds the first keychain item in a specified keychain that matches specified attributes.

Not recommended

```
OSStatus KCFindFirstItem (
   KCRef keychain,
   const KCAttributeList *attrList,
   KCSearchRef *search,
   KCItemRef *item
);
```

Parameters

keychain
A reference to the keychain that you wish to search. If you pass a locked keychain, the Unlock Keychain dialog box is displayed. If you pass `NULL`, the `KCFindFirstItem` function searches all unlocked keychains.

attrList
A pointer to a list of 0 or more structures containing information about the keychain item attributes to be matched. Pass `NULL` to match any attribute.

search
On return, a pointer to a reference to the current search criteria.

item
On return, a pointer to the first matching keychain item.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCItemNotFound` indicates that no matching keychain item was found. The result code `errKCNoSuchAttr` indicates that the specified attribute is undefined for this item class.

Discussion
The `KCFindFirstItem` function returns a reference to the first keychain item in a keychain that matches a list of attributes. The `KCFindFirstItem` function also returns a reference to the search criteria used. You should pass the returned search criteria in the `searchRef` parameter of the function `KCFindNextItem` (page 1148).

The `KCFindFirstItem` function automatically calls the function `KCUnlock` (page 1167) to display the Unlock Keychain dialog box if the keychain containing the item you are searching for is currently locked.

Special Considerations

When you are completely finished with a search, you should the functions `KCReleaseItem` (page 1161) and `KCReleaseSearch` (page 1162) to release the memory occupied by the keychain item and search criteria reference.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the Keychain Services function `SecKeychainSearchCreateFromAttributes` followed by a call to `SecKeychainSearchCopyNext` instead.

Declared In
`KeychainCore.h`

KCFindGenericPassword

Finds the first generic password in the default keychain matching the specified parameters.

Not recommended

```
OSStatus KCFindGenericPassword (
   ConstStringPtr serviceName,
   ConstStringPtr accountName,
   UInt32 maxLength,
   void *passwordData,
   UInt32 *actualLength,
   KCItemRef *item
);
```

Parameters

`serviceName`

A pointer to a Pascal string containing an application-defined service name. Pass `NULL` to match any service name.

accountName

A pointer to a Pascal string containing an application-defined account name. Pass `NULL` to match any account name.

maxLength

The length of the buffer pointed to by `passwordData`.

passwordData

A pointer to the buffer that holds the returned password data. Before calling the `KCFindGenericPassword` function, allocate enough memory for the buffer to hold the data you want to store. Pass `NULL` if you want to obtain the item reference but not the password data. In this case, you must also pass `NULL` in the `actualLength` parameter. On return, a pointer to the returned password data.

actualLength

On return, the actual length of the password data that was retrieved. If the buffer pointed to by the `passwordData` parameter is smaller than the actual length of the data, the `KCFindGenericPassword` function returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling the `KCFindGenericPassword` function again.

item

On return, a pointer to a reference to the found item. Pass `NULL` if you don't want to obtain this reference.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCNoDefaultKeychain` indicates that no default keychain was found. The result code `errKCItemNotFound` indicates that no matching password item was found. The result code `errKCBufferTooSmall` indicates that your application must allocate a new buffer of sufficient size before calling the function `KCFindGenericPassword` again.

Discussion

The `KCFindGenericPassword` function finds the first generic password item which matches the attributes you provide. The buffer specified in the `passwordData` parameter must be large enough to hold the password data, otherwise the function `KCFindGenericPassword` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling the function `KCFindGenericPassword` again. The `KCFindGenericPassword` function optionally returns a reference to the found item.

The `KCFindGenericPassword` function automatically calls the function `KCUnlock` (page 1167) to display the Unlock Keychain dialog box if the keychain containing the password is currently locked.

You can also call the function `kcfindgenericpassword` to find the first generic password matching specified attributes. The `kcfindgenericpassword` function requires that you pass a pointer to a C string instead of a pointer to a Pascal string for the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes

Available beginning with KeychainLib 1.0. In KeychainLib 1.0, the `kcfindgenericpassword` function provides the same functionality as the function `KCFindGenericPassword`, except that it accepts C strings rather than Pascal strings as arguments. In KeychainLib 2.0, you should use the `KCFindGenericPassword` function, since the `kcfindgenericpassword` function is provided for convenience only and may be removed from the header file at some point in the future.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainFindGenericPassword` function in Keychain Services instead.

Declared In

`KeychainCore.h`

kcfindgenericpassword

Not recommended

```
OSStatus kcfindgenericpassword (
   const char *serviceName,
   const char *accountName,
   UInt32 maxLength,
   void *passwordData,
   UInt32 *actualLength,
   KCItemRef *item
);
```

Discussion

This function is available for convenience only and may be removed. Use the function `KCFindGenericPassword` (page 1140) instead.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainFindGenericPassword` function in Keychain Services instead.

Declared In

`KeychainCore.h`

KCFindInternetPassword

Finds the first Internet password in the default keychain that matches the specified parameters.

Not recommended

```
OSStatus KCFindInternetPassword (
   ConstStringPtr serverName,
   ConstStringPtr securityDomain,
   ConstStringPtr accountName,
   UInt16 port,
   OSType protocol,
   OSType authType,
   UInt32 maxLength,
   void *passwordData,
   UInt32 *actualLength,
   KCItemRef *item
);
```

Parameters

serverName

A pointer to a Pascal string containing the server name. Pass `NULL` to match any server name.

securityDomain

A pointer to a Pascal string containing the security domain. Pass `NULL` to match any domain.

accountName

A pointer to a Pascal string containing the account name. Pass `NULL` to match any account name.

port

The TCP/IP port number. Pass the constant `kAnyPort`, described in "Default Internet Port Constant" (page 1180), to match any port.

protocol

The protocol associated with this password. See "Keychain Protocol Type Constants" (page 1191) for a description of possible values. Pass the constant `kAnyProtocol`, described in "Default Internet Protocol And Authentication Type Constants" (page 1180), to match any protocol.

authType

The authentication scheme used. See "Authentication Type Constants" (page 1176) for a description of possible values. Pass the constant `kAnyAuthType`, described in "Default Internet Protocol And Authentication Type Constants" (page 1180), to match any authentication scheme.

maxLength

The length of the buffer pointed to by `passwordData`.

passwordData

A pointer to the buffer that holds the returned password data. Before calling the `KCFindInternetPassword` function, allocate enough memory for the buffer to hold the data you want to store. Pass `NULL` if you want to obtain the item reference but not the password data. In this case, you must also pass `NULL` in the `actualLength` parameter. On return, a pointer to the returned password data.

actualLength

On return, the actual length of the password data that was retrieved. If the buffer pointed to by the `passwordData` parameter is smaller than the actual length of the data, the `KCFindInternetPassword` function returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling the `KCFindInternetPassword` function again.

item

On return, a pointer to a reference to the found item. Pass `NULL` if you don't want to obtain this reference.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCNoDefaultKeychain` indicates that no default keychain was found. The result code `errKCItemNotFound` indicates that no matching password item was found. The result code `errKCBufferTooSmall` indicates that your application must allocate a new buffer of sufficient size before calling the function `KCFindInternetPassword` again.

Discussion
The `KCFindInternetPassword` function finds the first Internet password item that matches the attributes you provide. The buffer specified in the `passwordData` parameter must be large enough to hold the password data, otherwise the function `KCFindInternetPassword` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling the function `KCFindInternetPassword` again. The `KCFindInternetPassword` function optionally returns a reference to the found item.

The `KCFindInternetPassword` function automatically calls the function `KCUnlock` (page 1167) to display the Unlock Keychain dialog box if the keychain containing the password is currently locked.

You can also call the function `kcfindinternetpassword` to find the first Internet password item matching specified attributes. The `kcfindinternetpassword` function requires that you pass a pointer to a C string instead of a Pascal string for the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes
Available beginning with KeychainLib 1.0. In KeychainLib 1.0, the `kcfindinternetpassword` function provides the same functionality as the function `KCFindInternetPassword`, except that it accepts C strings rather than Pascal strings as arguments. In KeychainLib 2.0, you should use the `KCFindInternetPassword` function, since `kcfindinternetpassword` is provided for convenience only and may be removed from the header file at some point in the future.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes
Use the `SecKeychainFindInternetPassword` function in Keychain Services instead.

Declared In
`KeychainCore.h`

kcfindinternetpassword
Not recommended

```
OSStatus kcfindinternetpassword (
   const char *serverName,
   const char *securityDomain,
   const char *accountName,
   UInt16 port,
   OSType protocol,
   OSType authType,
   UInt32 maxLength,
   void *passwordData,
   UInt32 *actualLength,
   KCItemRef *item
);
```

Discussion
This function is available for convenience only and may be removed. Use the function `KCFindInternetPassword` (page 1142) instead.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainFindInternetPassword` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCFindInternetPasswordWithPath

Finds the first Internet password in the default keychain that matches the specified parameters, including path information.

Not recommended

```
OSStatus KCFindInternetPasswordWithPath (
   ConstStringPtr serverName,
   ConstStringPtr securityDomain,
   ConstStringPtr accountName,
   ConstStringPtr path,
   UInt16 port,
   OSType protocol,
   OSType authType,
   UInt32 maxLength,
   void *passwordData,
   UInt32 *actualLength,
   KCItemRef *item
);
```

Parameters

serverName
A pointer to a Pascal string containing the server name. Pass `NULL` to match any server name.

securityDomain
A pointer to a Pascal string containing the security domain. Pass `NULL` to match any domain.

accountName

A pointer to a Pascal string containing the account name. Pass `NULL` to match any account name.

path

A pointer to a Pascal string containing additional information that specifies a file or directory on the server specified by the `serverName` parameter. In a typical URL, path information begins directly after the first slash ("/") character following the server name. This parameter is optional.

port

The TCP/IP port number. Pass the constant `kAnyPort`, described in "Default Internet Port Constant" (page 1180), to match any port.

protocol

The protocol associated with this password. See "Keychain Protocol Type Constants" (page 1191) for a description of possible values. Pass the constant `kAnyProtocol`, described in "Default Internet Protocol And Authentication Type Constants" (page 1180), to match any protocol.

authType

The authentication scheme used. See "Authentication Type Constants" (page 1176) for a description of possible values. Pass the constant `kAnyAuthType`, described in "Default Internet Protocol And Authentication Type Constants" (page 1180), to match any authentication scheme.

maxLength

The length of the buffer pointed to by `passwordData`.

passwordData

A pointer to a buffer which will hold the returned password data. Before calling the `KCFindInternetPasswordWithPath` function, allocate enough memory for the buffer to hold the data you want to store. Pass `NULL` if you want to obtain the item reference but not the password data. In this case, you must also pass `NULL` in the `actualLength` parameter. On return, a pointer to the returned password data.

actualLength

On return, the actual length of the password data that was retrieved. If the buffer pointed to by the `passwordData` parameter is smaller than the actual length of the data, the function `KCFindInternetPasswordWithPath` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling the function `KCFindInternetPasswordWithPath` again.

item

On return, a pointer to a reference to the found item. Pass `NULL` if you don't want to obtain this reference.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCNoDefaultKeychain` indicates that no default keychain was found. The result code `errKCItemNotFound` indicates that no matching password item was found. The result code `errKCBufferTooSmall` indicates that your application must allocate a new buffer of sufficient size before calling the function `KCFindInternetPasswordWithPath` again.

Discussion

The `KCFindInternetPasswordWithPath` function finds the first Internet password item which matches the attributes you provide, including path information. The buffer specified in the `passwordData` parameter must be large enough to hold the password data, otherwise the function `KCFindInternetPasswordWithPath` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling the `KCFindInternetPasswordWithPath` function again. The `KCFindInternetPasswordWithPath` function optionally returns a reference to the found item.

The KCFindInternetPasswordWithPath function automatically calls the function KCUnlock (page 1167) to display the Unlock Keychain dialog box if the keychain containing the password is currently locked.

You can also call the function kcfindinternetpasswordwithpath to find the first Internet password item matching specified attributes. The function kcfindinternetpasswordwithpath requires that you pass a pointer to a C string instead of a pointer to a Pascal string for the serverAddress, serverName, volumeName, accountName, and passwordData parameters.

Version Notes
Available beginning with KeychainLib 2.0. In KeychainLib 1.0, the kcfindinternetpassword function provides the same functionality as the function KCFindInternetPassword, except that it accepts C strings rather than Pascal strings as arguments. In KeychainLib 2.0, you should use KCFindInternetPassword, since kcfindinternetpassword is provided for convenience only and may be removed from the header file at some point in the future.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the SecKeychainFindInternetPassword function in Keychain Services instead.

Declared In
KeychainCore.h

kcfindinternetpasswordwithpath

Not recommended

```
OSStatus kcfindinternetpasswordwithpath (
   const char *serverName,
   const char *securityDomain,
   const char *accountName,
   const char *path,
   UInt16 port,
   OSType protocol,
   OSType authType,
   UInt32 maxLength,
   void *passwordData,
   UInt32 *actualLength,
   KCItemRef *item
);
```

Discussion
This function is available for convenience only and may be removed. Use the function KCFindInternetPasswordWithPath (page 1145) instead.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the SecKeychainFindInternetPassword function in Keychain Services instead.

Declared In
KeychainCore.h

KCFindNextItem

Finds the next keychain item matching the previously specified search criteria.

Not recommended

```
OSStatus KCFindNextItem (
   KCSearchRef search,
   KCItemRef *item
);
```

Parameters

search

A reference to the previously-specified search criteria. Pass the reference passed back in the searchRef parameter of the function KCFindFirstItem (page 1139).

item

On return, a pointer to the next matching keychain item, if any.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194). The result code errKCNoDefaultKeychain indicates that no default keychain could be found. The result code errKCItemNotFound indicates that no matching keychain item was found. The result code errKCInvalidSearchRef indicates that the specified search reference was invalid.

Discussion
The KCFindNextItem function finds the next keychain item matching the search criteria previously specified by a call to the function KCFindFirstItem (page 1139). The KCFindNextItem function returns a reference to the matching item, if any. The KCFindNextItem function automatically calls the function KCUnlock (page 1167) to display the Unlock Keychain dialog box if the keychain containing the item you are searching for is currently locked.

Special Considerations

When you are completely finished with a search, you should use the functions KCReleaseItem (page 1161) and KCReleaseSearch (page 1162) to release the keychain item and search criteria reference.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the SecKeychainSearchCopyNext function in Keychain Services instead.

Declared In
KeychainCore.h

KCFindX509Certificates

Finds the certificates in a keychain that match specified search criteria.

Unsupported

```
OSStatus KCFindX509Certificates (
    KCRef keychain,
    CFStringRef name,
    CFStringRef emailAddress,
    KCCertSearchOptions options,
    CFMutableArrayRef *certificateItems
);
```

Parameters

keychain
A reference to the keychain you want to search. If the keychain is locked, the Unlock Keychain dialog box is automatically displayed.

name
A pointer to a C string containing the certificate owner's common name.

emailAddress
A pointer to a C string containing the certificate owner's email address.

options
The search criteria you wish to use. See "Certificate Search Options" (page 1177) for a description of this mask.

certificateItems
On return, a pointer to a list of the matching certificates. Pass `NULL` if you don't want to obtain these references.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCNoDefaultKeychain` indicates that a default keychain was not found. The result code `errKCBufferTooSmall` indicates that the certificate data was too large for the supplied buffer. In this case, you should allocate a new buffer of sufficient size before calling `KCFindX509Certificates` again. The result code `errKCItemNotFound` indicates that no matching certificate was found.

Version Notes

Available beginning with KeychainLib 2.0.

Carbon Porting Notes

This function is obsolete. There is currently no replacement.

Declared In

`KeychainHI.h`

KCGetAttribute

Determines keychain item data using a keychain item attribute structure.

Not recommended

```
OSStatus KCGetAttribute (
   KCItemRef item,
   KCAttribute *attr,
   UInt32 *actualLength
);
```

Parameters

item

A reference to the keychain item whose attribute data you wish to determine.

attr

A pointer to a structure of type `KCAttribute` (page 1172). Before calling the `KCGetAttribute` function, fill in the `tag`, `length`, and `data` fields (the `data` field should contain a pointer to a buffer of sufficient length for the type of data to be returned). On return, the `KCGetAttribute` function passes back the requested data in the `data` field.

actualLength

On return, a pointer to the actual length of the attribute data. This may be more than the length you allocated in the length field of the attribute structure.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid. The result code `errKCNoSuchAttr` indicates that you tried to set an attribute which is undefined for this item class. The result code `errKCBufferTooSmall` indicates that your application must allocate a new buffer of sufficient size before calling `KCGetAttribute` again.

Discussion

You can call the function `KCGetAttribute` or the function `KCGetData` (page 1151) to obtain keychain item data. The difference between the functions is that the function `KCGetData` (page 1151) requires that you pass the length of the data and a pointer to that data as separate parameters rather than fields in a keychain item attribute structure.

If the keychain that contains the item is locked, before calling the `KCGetAttribute` function you should call the function `KCUnlock` (page 1167) to prompt the user to unlock the keychain.

You can determine any of the standard item attributes identified by the following tag constants: `kClassKCItemAttr`, `kCreationDateKCItemAttr`, `kModDateKCItemAttr`, `kDescriptionKCItemAttr`, `kCommentKCItemAttr`, `kLabelKCItemAttr`, `kCreatorKCItemAttr`, `kScriptCodeKCItemAttr`, and `kCustomIconKCItemAttr`. There is additional data you can determine, depending upon the type of keychain item whose data you wish to obtain. See "Keychain Item Attribute Tag Constants" (page 1184) for more information.

Version Notes

Available beginning with KeychainLib 1.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemCopyAttributesAndData` function in Keychain Services instead.

Declared In

`KeychainCore.h`

KCGetData

Determines keychain item data.

Not recommended

```
OSStatus KCGetData (
   KCItemRef item,
   UInt32 maxLength,
   void *data,
   UInt32 *actualLength
);
```

Parameters

item

A reference to the keychain item whose data you wish to determine.

maxLength

The length of the data buffer pointed to by the `data` parameter.

data

A pointer to the buffer that holds the returned data. Before calling the `KCGetData` function, allocate enough memory for the buffer to hold the data you want to store. On return, a pointer to the attribute data you requested.

actualLength

On return, a pointer to the actual length of the data being retrieved. If the buffer pointed to by the `data` parameter is smaller than the actual length of the data, the `KCGetData` function returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling `KCGetData` again.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid. The result code `errKCBufferTooSmall` indicates that your application must allocate a new buffer of sufficient size before calling `KCGetData` again. The result code `errKCDataNotModifiable` indicates that the data is not available for this item.

Discussion

You can call the function `KCGetData` or the function `KCGetAttribute` (page 1149) to obtain keychain item data. The difference between the functions is that the function `KCGetAttribute` (page 1149) requires that you pass the length of the data and a pointer to that data as fields in a keychain item attribute structure rather than as separate parameters.

If the keychain that contains the item is locked, before calling the function `KCGetData` you should call the function `KCUnlock` (page 1167) to prompt the user to unlock the keychain. You cannot call the `KCGetData` function for a private key.

You can determine any of the standard item attributes identified by the following tag constants: `kClassKCItemAttr`, `kCreationDateKCItemAttr`, `kModDateKCItemAttr`, `kDescriptionKCItemAttr`, `kCommentKCItemAttr`, `kLabelKCItemAttr`, `kCreatorKCItemAttr`, `kScriptCodeKCItemAttr`, and `kCustomIconKCItemAttr`. There is additional data you can determine, depending upon the type of keychain item whose data you wish to obtain. See "Keychain Item Attribute Tag Constants" (page 1184) for more information.

Version Notes

Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemCopyContent` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCGetDefaultKeychain

Obtains the default keychain.

Not recommended

```
OSStatus KCGetDefaultKeychain (
   KCRef *keychain
);
```

Parameters

keychain

On return, a pointer to default keychain reference.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCNoDefaultKeychain` indicates that there is no default keychain.

Discussion
You can determine the name of the default keychain by passing the returned keychain reference to the function `KCGetKeychainName` (page 1155).

Version Notes
Available beginning with KeychainLib 2.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainCopyDefault` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCGetIndKeychain

Obtains the reference to an indexed keychain.

Not recommended

```
OSStatus KCGetIndKeychain (
   UInt16 index,
   KCRef *keychain
);
```

Parameters

index

An index of the list of available keychains. Pass a value between 1 and the number returned by the function `KCCountKeychains` (page 1133).

keychain

On return, pointer to the keychain reference corresponding to the index in the `index` parameter.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCNoSuchKeychain` indicates that the index value is out of range.

Discussion

To guarantee correct operation, you should call the function `KCCountKeychains` (page 1133) once before calling `KCGetIndKeychain`.

Special Considerations

The memory that the keychain reference occupies must be released by calling the function `KCReleaseKeychain` (page 1162) when you are finished with it.

Version Notes

Available beginning with KeychainLib 1.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainCopySearchList` function in Keychain Services followed by a call to `CFArrayGetValueAtIndex` instead.

Declared In

`KeychainCore.h`

KCGetKeychain

Determines the location of a password or other keychain item.

Not recommended

```
OSStatus KCGetKeychain (
   KCItemRef item,
   KCRef *keychain
);
```

Parameters

item

A reference to the keychain item whose keychain location you wish to determine. If you pass a reference to a keychain item whose keychain is locked, the `KCGetKeychain` function returns the result code `errKCInvalidItemRef`.

keychain
On return, a pointer to the keychain containing the specified item.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCInvalidItemRef` indicates that the keychain item reference was invalid.

Discussion
The `KCGetKeychain` function determines the location of a keychain item in an unlocked keychain. It does not search locked keychains. Calling the `KCGetKeychain` function displays the Unlock Keychain dialog box if the keychain containing the item is currently locked.

Special Considerations

The keychain reference returned by `KCGetKeychain` should be released by calling the function `KCReleaseItem` (page 1161).

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemCopyKeychain` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCGetKeychainManagerVersion

Determines the version of the Keychain Manager installed on the user's system.

Not Recommended

```
OSStatus KCGetKeychainManagerVersion (
   UInt32 *returnVers
);
```

Parameters
returnVers
On return, a pointer to the version number of the Keychain Manager installed on the current system.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194).

Discussion
Your application can call the `KCGetKeychainManagerVersion` function to find out which version of the Keychain Manager is installed on the user's system.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainGetVersion` function in Keychain Services instead.

Declared In

`KeychainCore.h`

KCGetKeychainName

Determines the name of a keychain.

Not recommended

```
OSStatus KCGetKeychainName (
   KCRef keychain,
   StringPtr keychainName
);
```

Parameters

keychain

A reference to the keychain whose name you wish to obtain.

keychainName

A pointer to a Pascal string. On return, this string contains the name of the keychain.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCInvalidKeychain` indicates that the keychain is invalid.

Discussion

You can also call the function `kcgetkeychainname` to obtain the name of a keychain. `kcgetkeychainname` requires that you pass a pointer to a C string instead of a pointer to a Pascal string in the `keychainName` parameter.

Version Notes

Available beginning with KeychainLib 2.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainGetPath` function in Keychain Services instead.

Declared In

`KeychainCore.h`

kcgetkeychainname

Not recommended

```
OSStatus kcgetkeychainname (
   KCRef keychain,
   char *keychainName
);
```

Discussion
This function is available for convenience only and may be removed. Use the function `KCGetKeychainName` (page 1155) instead.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainGetPath` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCGetStatus

Determines the permissions that are set in a keychain.

Not recommended

```
OSStatus KCGetStatus (
   KCRef keychain,
   UInt32 *keychainStatus
);
```

Parameters

keychain

A pointer to the keychain reference whose permissions you wish to determine. Pass `NULL` to obtain the status of the default keychain.

keychainStatus

On return, a pointer to a bitmask that you can test to determine the permissions that are set in a keychain. See "Keychain Status Constants" (page 1193) for a description of this mask.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCNoSuchKeychain` indicates that the specified keychain could not be found. The result code `errKCInvalidKeychain` indicates that the specified keychain is invalid.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainGetStatus` function in Keychain Services instead.

Declared In
KeychainCore.h

KCIsInteractionAllowed

Indicates whether Keychain Manager functions that display a user interaction will do so.

Not recommended

```
Boolean KCIsInteractionAllowed (
   void
);
```

Parameters

Return Value
A Boolean value indicating whether user interaction is permitted. If true, user interaction is allowed, and Keychain Manager functions that display a user interface can do so as appropriate.

Version Notes
Available beginning with KeychainLib 2.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the SecKeychainGetUserInteractionAllowed function in Keychain Services instead.

Declared In
KeychainCore.h

KCLock

Locks a keychain.

Not recommended

```
OSStatus KCLock (
   KCRef keychain
);
```

Parameters

keychain

A reference to the keychain to lock. Pass NULL to lock all unlocked keychains.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194). The result code errKCNoSuchKeychain indicates that specified keychain could not be found. The result code errKCInvalidKeychain indicates that the specified keychain is invalid.

Discussion
Your application should not call the KCLock function unless you are responding to a user's request to lock a keychain. In general, you should leave the keychain unlocked so that the user does not have to unlock it again in another application.

Version Notes
The function `KCLock` replaces the function `KCLockKeychain`, which was available in KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainLock` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCMakeAliasFromKCRef

Creates an alias to a keychain reference.

Not Recommended

```
OSStatus KCMakeAliasFromKCRef (
   KCRef keychain,
   AliasHandle *keychainAlias
);
```

Parameters

keychain

A reference to the keychain for which you want to create an alias.

keychainAlias

On return, a pointer to an alias handle to the file referred to by the keychain reference.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194).

Discussion
You may wish to call the `KCMakeAliasFromKCRef` function to determine the location of a keychain.

Special Considerations

When you are finished with a keychain, you should call the function `KCReleaseKeychain` (page 1162) to deallocate its memory. You should not use the keychain after its memory has been deallocated.

Version Notes
Available beginning with KeychainLib 2.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainGetPath` function in Keychain Services followed by calls to the function `FSPathMakeRef` and `FSNewAlias` instead.

Declared In
`KeychainCore.h`

KCMakeKCRefFromAlias

Creates a keychain reference from a keychain alias.

Not Recommended

```
OSStatus KCMakeKCRefFromAlias (
   AliasHandle keychainAlias,
   KCRef *keychain
);
```

Parameters

keychainAlias

A handle to an alias record of the keychain file. Since the keychain is a file, an alias can be made to the keychain file.

keychain

On return, a pointer to a reference to the keychain specified by the alias in the `keychainAlias` parameter.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194).

Special Considerations

When you are finished with a keychain, you should call the function `KCReleaseKeychain` (page 1162) to deallocate its memory. You should not use the keychain after its memory has been deallocated.

Version Notes

Available beginning with KeychainLib 2.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainOpen` function in Keychain Services instead. If the keychain doesn't exist, use the `SecKeychainCreate` function in Keychain Services.

Declared In

`KeychainCore.h`

KCMakeKCRefFromFSSpec

Creates a keychain reference from a file specification record. (Deprecated in Mac OS X v10.5.)

Not Recommended

```
OSStatus KCMakeKCRefFromFSSpec (
   FSSpec *keychainFSSpec,
   KCRef *keychain
);
```

Parameters

keychainFSSpec

A pointer to a keychain file specification record.

keychain

On return, a pointer to a reference to the keychain specified by the file in the `keychainFSSpec` parameter.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194).

Special Considerations

When you are finished with a keychain, you should call the function `KCReleaseKeychain` (page 1162) to deallocate its memory. You should not use the keychain after its memory has been deallocated.

Version Notes
Available beginning with KeychainLib 2.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 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.

Carbon Porting Notes

Use the `SecKeychainOpen` function in Keychain Services instead. If the keychain doesn't exist, use the `SecKeychainCreate` function in Keychain Services.

Declared In
`KeychainCore.h`

KCNewItem

Creates a reference to a keychain item.

Not recommended

```
OSStatus KCNewItem (
   KCItemClass itemClass,
   OSType itemCreator,
   UInt32 length,
   const void *data,
   KCItemRef *item
);
```

Parameters

itemClass

The type of keychain item that you wish to create. See "Keychain Item Type Constants" (page 1190) for a description of possible values and a description of the `KCItemClass` data type.

itemCreator

The creator code of the application that owns this item.

length

The length of the data to be stored in this item.

data

A pointer to a buffer containing the data to be stored in this item. Before calling `KCNewItem`, allocate enough memory for the buffer to hold the data you want to store.

item

On return, a pointer to a reference to the newly-created item.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194). The Memory Manager result code `memFullErr` indicates that you did not allocate enough memory in the current heap to create the item.

Discussion
After calling the `KCNewItem` function, you should call the function `KCAddItem` (page 1130) if you wish to permanently store a password or other keychain item. Note that a copy of the data buffer pointed to by the `data` parameter is stored in the newly-created item.

Special Considerations

When you are done with a keychain item, you should call the function `KCReleaseItem` (page 1161) to release its memory. You should not use the item after its memory has been deallocated.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemCreateFromContent` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCReleaseItem

Disposes of the memory occupied by a keychain item reference.

Not recommended

```
OSStatus KCReleaseItem (
   KCItemRef *item
);
```

Parameters

item

A pointer to a keychain item reference. Pass the keychain item reference whose memory you want to release. On return, the reference is set to `NULL` and should not be used again.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194).

Discussion
You should call the `KCReleaseItem` function to release the memory occupied by a keychain item reference when you are finished with it.

Version Notes
Available beginning with KeychainLib 1.0

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `CFRelease` function instead.

Declared In
`KeychainCore.h`

KCReleaseKeychain

Disposes of the memory associated with a keychain reference.

Not recommended

```
OSStatus KCReleaseKeychain (
   KCRef *keychain
);
```

Parameters

keychain

A pointer to a keychain reference. Pass the keychain reference whose memory you want to release. On return, the reference is set to `NULL` and should not be used again.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194).

Discussion
You should call the `KCReleaseKeychain` function to release the memory occupied by a keychain reference when you are finished with it. You should not use the reference after it has been released.

Version Notes
Available beginning with KeychainLib 2.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `CFRelease` function instead.

Declared In
`KeychainCore.h`

KCReleaseSearch

Disposes of the memory occupied by a search criteria reference.

Not recommended

```
OSStatus KCReleaseSearch (
   KCSearchRef *search
);
```

Parameters

search

A pointer to a search criteria reference. Pass the search criteria reference whose memory you want to release. On return, the reference is set to `NULL` and should not be used again.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCInvalidSearchRef` indicates that the specified search reference was invalid.

Discussion

You should call the `KCReleaseSearch` function to release the memory occupied by a search criteria reference when you are completely finished with a search performed by calling the functions `KCFindFirstItem` (page 1139) or `KCFindNextItem` (page 1148).

Version Notes

Available beginning with KeychainLib 1.0

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `CFRelease` function instead.

Declared In

`KeychainCore.h`

KCRemoveCallback

Unregisters your keychain event callback function.

Not recommended

```
OSStatus KCRemoveCallback (
   KCCallbackUPP callbackProc
);
```

Parameters

callbackProc

A Universal Procedure Pointer (UPP) to your keychain event callback function that was previously registered with the function `KCAddCallback` (page 1123).

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCInvalidCallback` indicates that the callback function was not previously registered.

Discussion

After you pass a UPP to your keychain event callback function to the `KCRemoveCallback` function, it will no longer be called by the Keychain Manager.

Special Considerations

After calling `KCRemoveCallback`, you should call the function `DisposeKCCallbackUPP` (page 1119) to dispose of the UPP to your callback function.

Version Notes

Available beginning with KeychainLib 1.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainRemoveCalllback` function in Keychain Services instead.

Declared In

`KeychainCore.h`

KCSetAttribute

Sets or edits keychain item data using a keychain item attribute structure.

Not recommended

```
OSStatus KCSetAttribute (
   KCItemRef item,
   KCAttribute *attr
);
```

Parameters

item

A reference to the keychain item whose data you wish to set or edit.

attr

A pointer to a structure of type `KCAttribute` (page 1172) containing keychain item data you want to set. Before calling the function `KCSetAttribute`, fill in the `tag`, `length`, and `data` fields of this structure with the tag identifying the attribute you wish to modify or set, the length of the attribute data you wish to set, and a pointer to that data, respectively.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCInvalidItemRef` indicates that the keychain item reference was invalid. The result code `errKCNoSuchAttr` indicates that the item attribute you wish to set is undefined for the specified item. The result code `errKCDataTooLarge` indicates that more data was supplied than is allowed for this attribute.

Discussion

You can call the `KCSetAttribute` function or the function `KCSetData` (page 1165) to set or modify keychain item data. The difference between the functions is that the `KCSetData` (page 1165) function requires that you pass the length of the data and a pointer to that data as separate parameters rather than fields in a keychain item attribute structure.

If the keychain that contains the item is locked, before calling the `KCSetAttribute` function you should call the function `KCUnlock` (page 1167) to prompt the user to unlock the keychain. The keychain must permit read/write access in order to modify keychain item data.

You can only set or modify standard item attributes identified by the tag constants `kDescriptionKCItemAttr`, `kCommentKCItemAttr`, `kLabelKCItemAttr`, `kCreatorKCItemAttr`, `kTypeKCItemAttr`, and `kCustomIconKCItemAttr`. In addition, each class of keychain item has attributes specific to that class which may be set or modified. See "Keychain Item Attribute Tag Constants" (page 1184) for more information.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemModifyAttributesAndData` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCSetData

Sets or edits keychain item data.

Not recommended

```
OSStatus KCSetData (
   KCItemRef item,
   UInt32 length,
   const void *data
);
```

Parameters

item
A reference to the keychain item whose data you wish to set.

length
The length of the data buffer pointed to by the `data` parameter.

data
A pointer to a buffer containing the data to be stored in this item. Before calling the `KCSetData` function, allocate enough memory for the buffer to hold the data you want to store.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid. The result code `errKCDataTooLarge` indicates that the data was too large for the supplied buffer. The result code `errKCDataNotModifiable` indicates that the data cannot be set for this item.

Discussion
You can call the function `KCSetData` or the function `KCSetAttribute` (page 1164) to set or modify keychain item data. The difference between the functions is that the function `KCSetAttribute` (page 1164) requires that you pass the length of the data buffer as a field in a keychain item attribute structure rather than as a separate parameter.

If the keychain that contains the item is locked, before calling the `KCSetData` function you should call the function `KCUnlock` (page 1167) to prompt the user to unlock the keychain. The keychain must permit read/write access in order to modify keychain item data.

You can set or edit any of the standard item attributes identified by the following tag constants: `kDescriptionKCItemAttr`, `kCommentKCItemAttr`, `kLabelKCItemAttr`, `kCreatorKCItemAttr`, `kTypeKCItemAttr`, and `kCustomIconKCItemAttr`. There is additional data you can set, depending upon the type of keychain item whose data you are manipulating. See "Keychain Item Attribute Tag Constants" (page 1184) for more information.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemModifyContent` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCSetDefaultKeychain

Sets the default keychain.

Not recommended

```
OSStatus KCSetDefaultKeychain (
   KCRef keychain
);
```

Parameters

keychain

A reference to the keychain you wish to make the default.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194). The result code `errKCNoSuchKeychain` indicates that the specified keychain could not be found. The result code `errKCInvalidKeychain` indicates that the specified keychain is invalid.

Discussion
In most cases, your application should not need to set the default keychain, because this is a choice normally made by the user. You should call the `KCSetDefaultKeychain` function to change where a password or other keychain items are added.

The `KCSetDefaultKeychain` function sets the default keychain regardless of whether the keychain is currently locked.

Version Notes
Available beginning with KeychainLib 2.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainSetDefault` function in Keychain Services instead.

Declared In

`KeychainCore.h`

KCSetInteractionAllowed

Enables or disables Keychain Manager functions that display a user interface.

Not recommended

```
OSStatus KCSetInteractionAllowed (
   Boolean state
);
```

Parameters

state

A flag that indicates whether the Keychain Manager will display a user interface. If you pass `true`, user interaction is allowed. This is the default value. If `false`, Keychain Manager functions that normally display a user interface will instead return an error.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194).

Discussion

The `KCSetInteractionAllowed` function enables you to control whether the functions `KCLock` (page 1157), `KCUnlock` (page 1167), and `KCChangeSettings` (page 1131) display a user interface. Note that failure to re-enable user interaction will affect other clients of the Keychain Manager. By default, user interaction is permitted.

Version Notes

Available beginning with KeychainLib 2.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainSetUserInteractionAllowed` function in Keychain Services instead.

Declared In

`KeychainCore.h`

KCUnlock

Displays a dialog box that prompts the user for a password before unlocking a keychain.

Not recommended

```
OSStatus KCUnlock (
   KCRef keychain,
   StringPtr password
);
```

Parameters

keychain

A reference to the keychain to unlock. Pass `NULL` to specify the default keychain. If you pass `NULL` and the default keychain is currently locked, the keychain will appear as the default choice. If you pass a locked keychain, the function `KCUnlock` displays the Unlock Keychain dialog box and the keychain appears as the chosen menu item in the keychain popup menu. If the default keychain is currently unlocked, the Unlock Keychain dialog box is not displayed and the `KCUnlock` function returns `noErr`.

password

A pointer to a Pascal string representing the password string for this keychain. Pass `NULL` if the user password is unknown. In this case, The `KCUnlock` function displays the Unlock Keychain dialog box, and the authentication user interface associated with the keychain about to be unlocked. If you specify an invalid password, you will not be able to unlock the keychain with a specified password until the machine is rebooted. In this case, the `KCUnlock` function returns `errKCInteractionRequired`.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). The result code `noErr` does not guarantee that the specified keychain is unlocked, because the user can select any available keychain and unlock it. The result code `userCanceledErr` indicates that the user pressed the Cancel button in the Unlock Keychain dialog box. The result code `errKCAuthFailed` indicates that authentication failed because of too many unsuccessful retries. The result code `errKCInteractionRequired` indicates that user interaction is required to unlock the keychain. In this case, you will not be able to unlock the keychain with that password until the machine is rebooted.

Discussion

In most cases, your application does not need to call the `KCUnlock` function directly, since most Keychain Manager functions that require an unlocked keychain call `KCUnlock` automatically. If your application needs to verify that a keychain is unlocked, call the function `KCGetStatus` (page 1156).

You can also call the function `kcunlock` to display a user interface prompting the user to unlock a keychain. The `kcunlock` function requires that you pass a pointer to a C string instead of a pointer to a Pascal string in the `password` parameter.

Special Considerations

It is recommended that the `KCUnlock` function not be explicitly called by applications. Most functions that require an unlocked keychain call the `KCUnlock` function for you.

The memory that the keychain reference occupies must be released by calling the function `KCReleaseKeychain` (page 1162) when you are finished with it.

Version Notes

The `KCUnlock` function replaces the function `KCUnlockKeychain`, which was available in KeychainLib 1.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainUnlock` function in Keychain Services instead.

Declared In
KeychainHI.h

kcunlock

Not recommended

```
OSStatus kcunlock (
   KCRef keychain,
   const char *password
);
```

Discussion
This function is available for convenience only and may be removed. Use the function KCUnlock (page 1167) instead.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the SecKeychainUnlock function in Keychain Services instead.

Declared In
KeychainHI.h

KCUpdateItem

Updates a password or other keychain item.

Not recommended

```
OSStatus KCUpdateItem (
   KCItemRef item
);
```

Parameters

item

A reference to the keychain item whose data you wish to update. If you pass an item that has not been previously added to the keychain, the KCUpdateItem function does nothing and returns noErr.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1194). The result code errKCNoDefaultKeychain indicates that no default keychain could be found. The result code errKCInvalidItemRef indicates that the specified keychain item reference was invalid.

Discussion
You can use the KCUpdateItem function to update a password or other keychain item in a keychain's permanent data store after changing its data. The function KCUpdateItem automatically calls the function KCUnlock (page 1167) to display the Unlock Keychain dialog box if the keychain containing the item is currently locked.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemModifyContent` function in Keychain Services instead.

Declared In
`KeychainCore.h`

NewKCCallbackUPP

Creates a UPP to your keychain event callback.

Not recommended

```
KCCallbackUPP NewKCCallbackUPP (
   KCCallbackProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your keychain event callback function. For information on how to create a keychain event callback, see `KCCallbackProcPtr` (page 1171).

Return Value
A UPP to your callback function. You can register your callback function by passing this UPP in the `callbackProc` parameter of the function `KCAddCallback` (page 1123). See the description of the `KCCallbackUPP` data type.

Discussion
The `NewKCCallbackUPP` function creates a pointer to your keychain event callback function. You pass a pointer to your callback function in the `callbackProc` parameter of the function `KCAddCallback` (page 1123) if you want your application to receive data transfer events.

Special Considerations

When you are finished with a UPP to your keychain event callback function, you should dispose of it by calling the function `DisposeKCCallbackUPP` (page 1119).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

There is no replacement function available.

Declared In
`KeychainCore.h`

Callbacks

KCCallbackProcPtr

Defines a pointer to your keychain event callback that handles user keychain access events.

```
typedef OSStatus (*KCCallbackProcPtr)
(
        KCEvent keychainEvent,
        KCCallbackInfo * info,
        void * userContext
);
```

If you name your function `MyKCCallbackProc`, you would declare it like this:

```
OSStatus MyKCCallbackProc (
        KCEvent keychainEvent,
        KCCallbackInfo * info,
        void * userContext
);
```

Parameters

keychainEvent

The keychain event that your application wishes to be notified of. See "Keychain Events Constants" (page 1180) for a description of possible values. The type of event that can trigger your callback depends on the bitmask you passed in the `eventMask` parameter of the function `KCAddCallback` (page 1123). For more information, see the discussion.

info

A pointer to a structure of type `KCCallbackInfo` (page 1173). On return, the structure contains information about the keychain event that occurred. The Keychain Manager passes this information to your callback function via the `info` parameter of the function `InvokeKCCallbackUPP` (page 1119).

userContext

A pointer to application-defined storage that your application previously passed to the function `KCAddCallback` (page 1123). You can use this value to perform operations such as tracking which instance of a function is operating.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1194). Your keychain event callback function should process the keychain event and return `noErr`.

Discussion

Your keychain event callback function handles those keychain events that you indicate. In order to be notified of these events, you must pass a UPP to your notification callback function in the `callbackProc` parameter of `KCAddCallback` (page 1123). You indicate the type of data transfer events you want to receive via a bitmask in the `eventMask` parameter. When you no longer wish to receive notification of keychain events, you should call the function `KCRemoveCallback` (page 1163) to dispose of the UPP to your keychain event callback function.

Carbon Porting Notes

Use the `SecKeychainCallback` function in Keychain Services instead.

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

Declared In
`KeychainCore.h`

Data Types

AFPServerSignature

Represents a 16-byte Apple File Protocol server signature block.

```
typedef UInt8 AFPServerSignature[16];
```

Discussion
The `AFPServerSignature` type represents a 16-byte Apple File Protocol server signature block. You can pass a value of this type in the serverSignature parameter of the functions `KCAddAppleSharePassword` (page 1120) and `KCFindAppleSharePassword` (page 1136) to represent an Apple File Protocol server signature. You can use a value of this type with the keychain item attribute constant `kSignatureKCItemAttr` to specify an Apple File Protocol server signature.

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

Declared In
`KeychainCore.h`

KCAttribute

Contains information about a keychain item attribute.

```
typedef SecKeychainAttribute KCAttribute;
```

Discussion
The `KCAttribute` type represents a structure containing information about the attribute of a keychain item. It contains a tag that identifies a particular keychain item attribute value, the length of the attribute value, and a pointer to the attribute value. You can modify attribute data for a keychain item attribute by passing a pointer to this structure in the `attr` parameter of the function `KCSetAttribute` (page 1164). The function `KCGetAttribute` (page 1149) passes back a pointer to this structure in the `attr` parameter.

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

Declared In
`KeychainCore.h`

KCAttributeList

Lists attributes in a keychain item.

```
typedef SecKeychainAttributeList KCAttributeList;
```

Discussion
The `KCAttributeList` type represents a list of structures containing information about the attributes in a keychain item. You pass a pointer to this list of 0 or more structures in the `attrList` parameter of the function `KCFindFirstItem` (page 1139) to indicate the attributes to be matched.

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

Declared In
`KeychainCore.h`

KCAttrType

Identifies a keychain item attribute value.

```
typedef SecKeychainAttrType KCAttrType;
```

Discussion
The `KCAttrType` type represents a tag that identifies a keychain item attribute value. You can use this value in the `tag` field of the structure `KCAttribute` (page 1172) to identify the keychain item attribute value you wish to set or obtain. See `Keychain Item Attribute Tag Constants` (page 1184) for a description of the Apple-defined tag constants and the data types of the values they identify. Your application can create application-defined tags of type `KCAttrType`.

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

Declared In
`KeychainCore.h`

KCCallbackInfo

Contains information about a keychain event.

```
struct KCCallbackInfo {
    UInt32 version;
    KCItemRef item;
    long processID[2];
    long event[4];
    KCRef keychain;
};
typedef struct KCCallbackInfo KCCallbackInfo;
```

Fields

`version`
The version of this structure.

`item`
A reference to the keychain item in which the event occurred. If the event did not involve an item, this field is not valid.

`processID`
: A 64-bit quantity containing the process serial number of the process in which the event occurred. This is not available on Mac OS X.

`event`
: The keychain event that occurred. If the event is a system event as indicated by the constant `kSystemKCEvent`, the Keychain client can process events. If the event is not a system event, this field is not valid. This is not available on Mac OS X.

`keychain`
: A reference to the keychain in which the event occurred. If the event did not involve a keychain, this field is not valid.

Discussion
The `KCCallbackInfo` type represents a structure that contains information about the keychain event of which your application wants to be notified. The Keychain Manager passes a pointer to this structure in the `info` parameter of your callback function via the function `InvokeKCCallbackUPP` (page 1119), which invokes your callback function. For information on how to write a keychain event callback function, see `KCCallbackProcPtr` (page 1171).

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

Declared In
`KeychainCore.h`

KCCallbackUPP

Defines a data type for the `KCCallbackProcPtr` callback pointer.

```
typedef KCCallbackProcPtr KCCallbackUPP;
```

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

Declared In
`KeychainCore.h`

KCItemRef

Represents a reference to a keychain item.

```
typedef SecKeychainItemRef KCItemRef;
```

Discussion
The `KCItemRef` type represents a reference to an opaque structure that identifies a keychain item. You should call the function `KCNewItem` (page 1160) to create a keychain item reference. The function `KCReleaseItem` (page 1161) disposes of a keychain item reference when no longer needed.

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

Declared In
`KeychainCore.h`

KCPublicKeyHash

Represents a 20-byte public key hash.

```
typedef UInt8 KCPublicKeyHash[20];
```

Discussion
The `KCPublicKeyHash` type represents a hash of a public key. You can use the constant `kPublicKeyHashKCItemAttr`, described in `Keychain Item Attribute Tag Constants` (page 1184), to set or retrieve a certificate attribute value of this type.

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

Declared In
`KeychainCore.h`

KCRef

Represents a reference to a keychain.

```
typedef SecKeychainRef KCRef;
```

Discussion
The `KCRef` type represents a reference to an opaque structure that identifies a keychain. You should call the function `KCMakeKCRefFromFSSpec` (page 1159) or `KCMakeKCRefFromAlias` (page 1159) to create a keychain reference. The function `KCReleaseKeychain` (page 1162) disposes of a keychain reference when no longer needed. You pass a reference of this type to Keychain Manager functions that operate on a keychain in some way.

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

Declared In
`KeychainCore.h`

KCSearchRef

Represents a reference to the current search criteria.

```
typedef SecKeychainSearchRef KCSearchRef;
```

Discussion
The `KCSearchRef` type represents a reference to an opaque structure that identifies the current search criteria. The function `KCFindFirstItem` (page 1139) passes back a reference of this type in the `search` parameter for subsequent calls to the function `KCFindNextItem` (page 1148). You must release this reference when you are finished with a search by calling the function `KCReleaseSearch` (page 1162).

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

Declared In
`KeychainCore.h`

KCStatus

Identifies a mask that you can use in determining the permissions that are set in a keychain.

```
typedef SecKeychainStatus KCStatus;
```

Discussion
The `KCStatus` enumeration defines masks your application can use to determine the read and write permissions for a keychain. The function `KCGetStatus` (page 1156) passes back this mask in the `status` parameter.

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

Declared In
`KeychainCore.h`

Constants

Authentication Type Constants

Represent the type of authentication to use in storing and retrieving Internet passwords.

```
enum {
    kKCAuthTypeNTLM = 'ntlm',
    kKCAuthTypeMSN = 'msna',
    kKCAuthTypeDPA = 'dpaa',
    kKCAuthTypeRPA = 'rpaa',
    kKCAuthTypeHTTPDigest = 'httd',
    kKCAuthTypeDefault = 'dflt'
};
typedef FourCharCode KCAuthType;
```

Constants

`kKCAuthTypeNTLM`
Specifies Windows NT LAN Manager authentication.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCAuthTypeMSN`
Specifies Microsoft Network authentication.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCAuthTypeDPA`
Specifies Distributed Password authentication.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCAuthTypeRPA`
Specifies Remote Password authentication.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kKCAuthTypeHTTPDigest`
Specifies HTTP Digest Access authentication.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kKCAuthTypeDefault`
Specifies default authentication.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

Discussion
The `KCAuthType` enumeration defines constants you can use to identify the type of authentication to use in storing and retrieving Internet passwords. You can pass a constant of this type in the `authType` parameter of the functions `KCAddInternetPassword` (page 1126), `KCAddInternetPasswordWithPath` (page 1128), `KCFindInternetPassword` (page 1142), and `KCFindInternetPasswordWithPath` (page 1145).

Certificate Search Options

Represent a mask that specifies the search criteria to use when finding certificates.

```
typedef UInt32 KCCertSearchOptions;
enum {
    kCertSearchShift = 0,
    kCertSearchSigningIgnored = 0,
    kCertSearchSigningAllowed = 1 << (kCertSearchShift + 0),
    kCertSearchSigningDisallowed = 1 << (kCertSearchShift + 1),
    kCertSearchSigningMask = ((kCertSearchSigningAllowed) |
        (kCertSearchSigningDisallowed)),
    kCertSearchVerifyIgnored = 0,
    kCertSearchVerifyAllowed = 1 << (kCertSearchShift + 2),
    kCertSearchVerifyDisallowed = 1 << (kCertSearchShift + 3),
    kCertSearchVerifyMask = ((kCertSearchVerifyAllowed) |
        (kCertSearchVerifyDisallowed)),
    kCertSearchEncryptIgnored = 0,
    kCertSearchEncryptAllowed = 1 << (kCertSearchShift + 4),
    kCertSearchEncryptDisallowed = 1 << (kCertSearchShift + 5),
    kCertSearchEncryptMask = ((kCertSearchEncryptAllowed) |
        (kCertSearchEncryptDisallowed)),
    kCertSearchDecryptIgnored = 0,
    kCertSearchDecryptAllowed = 1 << (kCertSearchShift + 6),
    kCertSearchDecryptDisallowed = 1 << (kCertSearchShift + 7),
    kCertSearchDecryptMask = ((kCertSearchDecryptAllowed) |
        (kCertSearchDecryptDisallowed)),
    kCertSearchWrapIgnored = 0,
    kCertSearchWrapAllowed = 1 << (kCertSearchShift + 8),
    kCertSearchWrapDisallowed = 1 << (kCertSearchShift + 9),
    kCertSearchWrapMask = ((kCertSearchWrapAllowed) |
        (kCertSearchWrapDisallowed)),
    kCertSearchUnwrapIgnored = 0,
    kCertSearchUnwrapAllowed = 1 << (kCertSearchShift + 10),
    kCertSearchUnwrapDisallowed = 1 << (kCertSearchShift + 11),
    kCertSearchUnwrapMask = ((kCertSearchUnwrapAllowed) |
        (kCertSearchUnwrapDisallowed)),
    kCertSearchPrivKeyRequired = 1 << (kCertSearchShift + 12),
    kCertSearchAny = 0
};
```

Discussion
The `KCCertSearchOptions` enumeration defines masks that you can use in the `options` parameter of the function `KCFindX509Certificates` (page 1149).

Certificate Usage Options

Represent a mask that specifies the usage options when adding certificates.

```
typedef UInt32 KCCertAddOptions;
enum {
    kSecOptionReserved = 0x000000FF,
    kCertUsageShift = 8,
    kCertUsageSigningAdd         = 1 << (kCertUsageShift + 0),
    kCertUsageSigningAskAndAdd = 1 << (kCertUsageShift + 1),
```

```
    kCertUsageVerifyAdd = 1 << (kCertUsageShift + 2),
    kCertUsageVerifyAskAndAdd = 1 << (kCertUsageShift + 3),
    kCertUsageEncryptAdd = 1 <<(kCertUsageShift + 4),
    kCertUsageEncryptAskAndAdd = 1 << (kCertUsageShift + 5),
    kCertUsageDecryptAdd = 1 << (kCertUsageShift + 6),
    kCertUsageDecryptAskAndAdd = 1 << (kCertUsageShift + 7),
    kCertUsageKeyExchAdd = 1 << (kCertUsageShift + 8),
    kCertUsageKeyExchAskAndAdd = 1 << (kCertUsageShift + 9),
    kCertUsageRootAdd = 1 << (kCertUsageShift + 10),
    kCertUsageRootAskAndAdd = 1 << (kCertUsageShift + 11),
    kCertUsageSSLAdd = 1 << (kCertUsageShift + 12),
    kCertUsageSSLAskAndAdd = 1 << (kCertUsageShift + 13),
    kCertUsageAllAdd = 0x7FFFFF00
};
```

Certificate Verification Criteria

Identify the verification criteria for use when displaying certificates to the user.

```
typedef UInt16 KCVerifyStopOn;
enum {
    kPolicyKCStopOn = 0,
    kNoneKCStopOn = 1,
    kFirstPassKCStopOn = 2,
    kFirstFailKCStopOn = 3
};
```

Constants

`kPolicyKCStopOn`
Indicates that the function `KCChooseCertificate` (page 1132) should use the trust policy options currently in effect.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kNoneKCStopOn`
Indicates that the function `KCChooseCertificate` (page 1132) completes after examining all available certificates.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kFirstPassKCStopOn`
Indicates that the function `KCChooseCertificate` (page 1132) when one certificate meeting the verification criteria is found.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kFirstFailKCStopOn`
Specifies that the function `KCChooseCertificate` (page 1132) completes when one certificate that fails to meet the verification criteria is found.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

Discussion
The `KCVerifyStopOn` enumeration defines constants your application can use to identify the verification criteria to use in selecting certificates. You can pass a constant of this type in the `stopOn` parameter of the function `KCChooseCertificate` (page 1132).

Default Internet Port Constant

Represent the internet ports available.

```
enum {
    kAnyPort = 0
};
```

Constants

`kAnyPort`
Indicates that any Internet port can be used.
Available in Mac OS X v10.1 and later.
Declared in `KeychainCore.h`.

Default Internet Protocol And Authentication Type Constants

Represent the internet protocols and authentication types available.

```
enum {
    kAnyProtocol = 0,
    kAnyAuthType = 0
};
```

Constants

`kAnyProtocol`
Indicates that any Internet protocol can be used.
Available in Mac OS X v10.1 and later.
Declared in `KeychainCore.h`.

`kAnyAuthType`
Indicates that any Internet authentication type can be used.
Available in Mac OS X v10.1 and later.
Declared in `KeychainCore.h`.

Keychain Events Constants

Identify keychain events.

```
typedef UInt16 KCEvent;
enum {
    kIdleKCEvent = 0,
    kLockKCEvent = 1,
    kUnlockKCEvent = 2,
    kAddKCEvent = 3,
    kDeleteKCEvent = 4,
    kUpdateKCEvent = 5,
    kPasswordChangedKCEvent = 6,
    kSystemKCEvent = 8,
    kDefaultChangedKCEvent = 9,
    kDataAccessKCEvent = 10,
    kKeychainListChangedKCEvent = 11
};
```

Constants

`kIdleKCEvent`
Indicates a `NULL` event.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kLockKCEvent`
Indicates that the keychain was locked.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kUnlockKCEvent`
Indicates that the keychain was unlocked.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kAddKCEvent`
Indicates that an item was added to a keychain.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kDeleteKCEvent`
Indicates that an item was deleted from a keychain.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kUpdateKCEvent`
Indicates that a keychain item was updated.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kPasswordChangedKCEvent`
Indicates that the identity of the keychain was changed.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kSystemKCEvent`
Indicates that the keychain client can process events.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kDefaultChangedKCEvent`
Indicates that the default keychain has changed.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kDataAccessKCEvent`
Indicates that a process has called the function `KCGetData` (page 1151) to access a keychain item's data.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kKeychainListChangedKCEvent`
Indicates that the list of keychains has changed.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

Discussion
The `KCEvent` enumeration defines constants that identify the Keychain-related events your callback function wishes to receive. The Keychain Manager tests a mask that you pass in the `eventMask` parameter of the function `KCAddCallback` (page 1123) to determine the data transfer events your notification callback function wishes to receive. It passes these events in the `keychainEvent` parameter of the function `InvokeKCCallbackUPP` (page 1119). For a description of the Keychain-related event masks, see `Keychain Events Mask` (page 1182).

Keychain Events Mask

Identify a mask that you can use to set the keychain events you wish to receive.

```
typedef UInt16 KCEventMask;
enum {
    kIdleKCEventMask = 1 << kIdleKCEvent,
    kLockKCEventMask = 1 << kLockKCEvent,
    kUnlockKCEventMask = 1 << kUnlockKCEvent,
    kAddKCEventMask = 1 << kAddKCEvent,
    kDeleteKCEventMask = 1 << kDeleteKCEvent,
    kUpdateKCEventMask = 1 << kUpdateKCEvent,
    kPasswordChangedKCEventMask = 1 << kPasswordChangedKCEvent,
    kSystemEventKCEventMask = 1 << kSystemKCEvent,
    kDefaultChangedKCEventMask = 1 << kDefaultChangedKCEvent,
    kDataAccessKCEventMask = 1 << kDataAccessKCEvent,
    kEveryKCEventMask = 0xFFFF
};
```

Constants
`kIdleKCEventMask`
If the bit specified by this mask is set, your callback function will be invoked during a `NULL` event.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kLockKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when the keychain is locked.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kUnlockKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when the keychain is unlocked.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kAddKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when an item is added to the keychain.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kDeleteKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when an item is removed from the keychain.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kUpdateKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when a keychain item is updated.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kPasswordChangedKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when the keychain identity is changed.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kSystemEventKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when the keychain client processes an event.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kDefaultChangedKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when the default keychain is changed.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kDataAccessKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when a process calls the function `KCGetData` (page 1151).

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kEveryKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when any of the above Keychain-related events occur.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

Discussion

The `KCEventMask` enumeration defines masks your application can use to set Keychain event bits. You pass this mask in the `eventMask` parameter of the function `KCAddCallback` (page 1123), thereby defining the Keychain-related events to which your callback will respond. The Keychain Manager uses this mask to test which events your callback function will handle. It passes these events in the `keychainEvent` parameter of the function `InvokeKCCallbackUPP` (page 1119). For a description of Keychain-related events, see `Keychain Events Constants` (page 1180).

Keychain Item Attribute Tag Constants

Represent tags that identify keychain item attribute values.

```
enum {
    kClassKCItemAttr = 'clas',
    kCreationDateKCItemAttr = 'cdat',
    kModDateKCItemAttr = 'mdat',
    kDescriptionKCItemAttr = 'desc',
    kCommentKCItemAttr = 'icmt',
    kCreatorKCItemAttr = 'crtr',
    kTypeKCItemAttr = 'type',
    kScriptCodeKCItemAttr = 'scrp',
    kLabelKCItemAttr = 'labl',
    kInvisibleKCItemAttr = 'invi',
    kNegativeKCItemAttr = 'nega',
    kCustomIconKCItemAttr = 'cusi',
    kAccountKCItemAttr = 'acct',
    kServiceKCItemAttr = 'svce',
    kGenericKCItemAttr = 'gena',
    kSecurityDomainKCItemAttr = 'sdmn',
    kServerKCItemAttr = 'srvr',
    kAuthTypeKCItemAttr = 'atyp',
    kPortKCItemAttr = 'port',
    kPathKCItemAttr = 'path',
    kVolumeKCItemAttr = 'vlme',
    kAddressKCItemAttr = 'addr',
    kSignatureKCItemAttr = 'ssig',
    kProtocolKCItemAttr = 'ptcl',
    kSubjectKCItemAttr = 'subj',
    kCommonNameKCItemAttr = 'cn  ',
    kIssuerKCItemAttr = 'issu',
    kSerialNumberKCItemAttr = 'snbr',
    kEMailKCItemAttr = 'mail',
    kPublicKeyHashKCItemAttr = 'hpky',
    kIssuerURLKCItemAttr = 'iurl',
    kEncryptKCItemAttr = 'encr',
    kDecryptKCItemAttr = 'decr',
    kSignKCItemAttr = 'sign',
    kVerifyKCItemAttr = 'veri',
    kWrapKCItemAttr = 'wrap',
    kUnwrapKCItemAttr = 'unwr',
    kStartDateKCItemAttr = 'sdat',
    kEndDateKCItemAttr = 'edat'
};
typedef FourCharCode KCItemAttr;
```

Constants

`kClassKCItemAttr`

Identifies the class attribute. You use this tag to set or get a value of type `KCItemClass` that indicates whether the item is an AppleShare, Internet, or generic password, or a certificate. See "KCPublicKeyHash" (page 1175) for a description of possible values.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kCreationDateKCItemAttr`

Identifies the creation date attribute. You use this tag to set or get a value of type `UInt32` that indicates the date the item was created.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kModDateKCItemAttr`
Identifies the modification date attribute. You use this tag to set or get a value of type `UInt32` that indicates the last time the item was updated.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kDescriptionKCItemAttr`
Identifies the description attribute. You use this tag to set or get a value of type `string` that represents a user-visible string describing this item.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kCommentKCItemAttr`
Identifies the comment attribute. You use this tag to set or get a value of type `string` that represents a user-editable string containing comments for this item.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kCreatorKCItemAttr`
Identifies the creator attribute. You use this tag to set or get a value of type `OSType` that represents the item's creator.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kTypeKCItemAttr`
Identifies the type attribute. You use this tag to set or get a value of type `OSType` that represents the item's type.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kScriptCodeKCItemAttr`
Identifies the script code attribute. You use this tag to set or get a value of type `ScriptCode` that represents the script code for all strings.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kLabelKCItemAttr`
Identifies the label attribute. You use this tag to set or get a value of type `string` that represents a user-editable string containing the label for this item.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kInvisibleKCItemAttr`
Identifies the invisible attribute. You use this tag to set or get a value of type `Boolean` that indicates whether the item is invisible.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kNegativeKCItemAttr`

Identifies the negative attribute. You use this tag to set or get a value of type `Boolean` that indicates whether there is a valid password associated with this keychain item. This is useful if your application doesn't want a password for some particular service to be stored in the keychain, but prefers that it always be entered by the user. The item (typically invisible and with zero-length data) acts as a placeholder to say “don't use me.”

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kCustomIconKCItemAttr`

Identifies the custom icon attribute. You use this tag to set or get a value of type `Boolean` that indicates whether the item has an application-specific icon. To do this, you must also set the attribute value identified by the tag kTypeKCItemAttr to a file type for which there is a corresponding icon in the desktop database, and set the attribute value identified by the tag `kCreatorKCItemAttr` to an appropriate application creator type. If a custom icon corresponding to the item's type and creator can be found in the desktop database, it will be displayed by Keychain Access. Otherwise, default icons are used.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kAccountKCItemAttr`

Identifies the account attribute. You use this tag to set or get a value of type `Str63` that represents the user account. It also applies to generic and AppleShare passwords.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kServiceKCItemAttr`

Identifies the service attribute for a generic password. You use this tag to set or get a value of type `Str63` that represents the service.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kGenericKCItemAttr`

Identifies the generic attribute for a generic password. You use this tag to set or get a value of untyped bytes that represents a user-defined attribute.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kSecurityDomainKCItemAttr`

Identifies the security domain attribute for an internet password. You use this tag to set or get a value of type `Str63` that represents the Internet security domain.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kServerKCItemAttr`

Identifies the server attribute for an internet password. You use this tag to set or get a value of type `string` that represents the Internet server's domain name or IP address.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kAuthTypeKCItemAttr`
Identifies the authentication type attribute for an internet password. You use this tag to set or get a value of type `KCAuthType` that represents the Internet authentication scheme.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kPortKCItemAttr`
Identifies the port attribute for an internet password. You use this tag to set or get a value of type `UInt16` that represents the Internet port.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kPathKCItemAttr`
Identifies the path attribute for an internet password. You use this tag to set or get a value of type `Str255` that represents the path.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kVolumeKCItemAttr`
Identifies the volume attribute for an AppleShare password. You use this tag to set or get a value of type `Str63` that represents the AppleShare volume.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kAddressKCItemAttr`
Identifies the address attribute for an AppleShare password. You use this tag to set or get a value of type `string` that represents the zone name, or the IP or domain name that represents the server address.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kSignatureKCItemAttr`
Identifies the server signature attribute for an AppleShare password. You use this tag to set or get a value of type `KCPublicKeyHash` (page 1175) that represents the server signature block.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kProtocolKCItemAttr`
Identifies the protocol attribute for an AppleShare or internet password. You use this tag to set or get a value of type `KCProtocolType` that represents the Internet protocol.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kSubjectKCItemAttr`
Identifies the subject attribute for a certificate. You use this tag to set or get `DER`-encoded data that represents the subject distinguished name.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kCommonNameKCItemAttr`
Identifies the common name attribute for a certificate. You use this tag to set or get a `UTF8`-encoded string that represents the common name.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kIssuerKCItemAttr`
Identifies the issuer attribute for a certificate. You use this tag to set or get a `DER`-encoded data that represents the issuer distinguished name.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kSerialNumberKCItemAttr`
Identifies the serial number attribute for a certificate. You use this tag to set or get a `DER`-encoded data that represents the serial number.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kEMailKCItemAttr`
Identifies the email attribute for a certificate. You use this tag to set or get an `ASCII`-encoded string that represents the issuer's email address.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kPublicKeyHashKCItemAttr`
Identifies the public key hash attribute for a certificate. You use this tag to set or get a value of type `KCPublicKeyHash` (page 1175) that represents the hash of the public key.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kIssuerURLKCItemAttr`
Identifies the issuer URL attribute for a certificate. You use this tag to set or get an `ASCII`-encoded string that represents the URL of the certificate issuer.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kEncryptKCItemAttr`
Identifies the encrypt attribute for a certificate or key. You use this tag to set or get a value of type `Boolean` that indicates whether the item can encrypt.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kDecryptKCItemAttr`
Identifies the decrypt attribute for a certificate or key. You use this tag to set or get a value of type `Boolean` that indicates whether the item can decrypt.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kSignKCItemAttr`
Identifies the sign attribute for a certificate or key. You use this tag to set or get a value of type `Boolean` that indicates whether the item can sign.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kVerifyKCItemAttr`
Identifies the verify attribute for a certificate or key. You use this tag to set or get a value of type `Boolean` that indicates whether the item can verify.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kWrapKCItemAttr`
Identifies the wrap attribute for a certificate or key. You use this tag to set or get a value of type `Boolean` that indicates whether the item can wrap.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kUnwrapKCItemAttr`
Identifies the unwrap attribute for a certificate or key. You use this tag to set or get a value of type `Boolean` that indicates whether the item can unwrap.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kStartDateKCItemAttr`
Identifies the start date attribute for a certificate or key. You use this tag to set or get a value of type `UInt32` that indicates the start date.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kEndDateKCItemAttr`
Identifies the end date attribute for a certificate or key. You use this tag to set or get a value of type `UInt32` that indicates the end date.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

Discussion
The `KCItemAttr` enumeration defines the Apple-defined tag constants that identify keychain item attribute values. Your application can use one of these tags in the `tag` field of the structure `KCAttribute` (page 1172) to identify the keychain item attribute value you wish to set or retrieve. Your application can create application-defined tags of type `KCAttrType` (page 1173).

Keychain Item Type Constants

Identify the type of keychain item.

```
enum {
    kCertificateKCItemClass = 'cert',
    kAppleSharePasswordKCItemClass = 'ashp',
    kInternetPasswordKCItemClass = 'inet',
    kGenericPasswordKCItemClass = 'genp'
};
typedef FourCharCode KCItemClass;
```

Constants

`kCertificateKCItemClass`
Specifies that the item is a digital certificate.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kAppleSharePasswordKCItemClass`
Specifies that the item is an AppleShare password.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kInternetPasswordKCItemClass`
Specifies that the item is an Internet password.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kGenericPasswordKCItemClass`
Specifies that the item is a generic password.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

Discussion

The `KCItemClass` enumeration defines constants your application can use to specify the type of the keychain item you wish to create, dispose, add, delete, update, copy, or locate. You pass a constant of this type to the functions `KCNewItem` (page 1160), `KCReleaseItem` (page 1161), `KCAddItem` (page 1130), `KCDeleteItem` (page 1136), `KCUpdateItem` (page 1169), `KCCopyItem` (page 1132), and `KCGetKeychain` (page 1153). You can also use these constants with the tag constant `kClassKCItemAttr`, described in `Keychain Item Attribute Tag Constants` (page 1184).

Keychain Protocol Type Constants

Identify the protocol to use in storing and retrieving Internet passwords.

```
enum {
    kKCProtocolTypeFTP = 'ftp ',
    kKCProtocolTypeFTPAccount = 'ftpa',
    kKCProtocolTypeHTTP = 'http',
    kKCProtocolTypeIRC = 'irc ',
    kKCProtocolTypeNNTP = 'nntp',
    kKCProtocolTypePOP3 = 'pop3',
    kKCProtocolTypeSMTP = 'smtp',
    kKCProtocolTypeSOCKS = 'sox ',
    kKCProtocolTypeIMAP = 'imap',
    kKCProtocolTypeLDAP = 'ldap',
    kKCProtocolTypeAppleTalk = 'atlk',
    kKCProtocolTypeAFP = 'afp ',
    kKCProtocolTypeTelnet = 'teln'
};
typedef FourCharCode KCProtocolType;
```

Constants

`kKCProtocolTypeFTP`
Specifies the File Transfer Protocol.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCProtocolTypeFTPAccount`
Specifies the File Transfer Protocol Account.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCProtocolTypeHTTP`
Specifies the HyperText Transfer Protocol.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCProtocolTypeIRC`
Specifies the Internet Relay Channel Protocol.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCProtocolTypeNNTP`
Specifies the Network News Transfer Protocol.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCProtocolTypePOP3`
Specifies the Post Office 3 Protocol.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCProtocolTypeSMTP`
Specifies the Simple Mail Transfer Protocol.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCProtocolTypeSOCKS`
Specifies the Secure Proxy Server Protocol.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kKCProtocolTypeIMAP`
Specifies the Internet Message Access Protocol.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kKCProtocolTypeLDAP`
Specifies the Lightweight Directory Access Protocol.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kKCProtocolTypeAppleTalk`
Specifies the AppleTalk Protocol.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kKCProtocolTypeAFP`
Specifies the AppleTalk File Protocol.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kKCProtocolTypeTelnet`
Specifies the Telnet Protocol.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

Discussion
The `KCProtocolType` enumeration defines constants you can use to identify the type of authentication to use in storing and retrieving Internet passwords. You can pass a constant of this type in the `protocol` parameter of the functions `KCAddInternetPassword` (page 1126), `KCAddInternetPasswordWithPath` (page 1128), `KCFindInternetPassword` (page 1142), and `KCFindInternetPasswordWithPath` (page 1145).

Keychain Status Constants

Identify the keychain status.

```
enum {
    kUnlockStateKCStatus = 1,
    kRdPermKCStatus = 2,
    kWrPermKCStatus = 4
};
```

Constants
`kUnlockStateKCStatus`
If the bit specified by this mask is set (bit 0), the keychain is unlocked.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kRdPermKCStatus`
If the bit specified by this mask is set (bit 1), the keychain is unlocked with read permission.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kWrPermKCStatus`
If the bit specified by this mask is set (bit 2), the keychain is unlocked with write permission.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

Result Codes

The most common result codes returned by Keychain Manager are listed below.

Result Code	Value	Description
`errKCNotAvailable`	-25291	Indicates that the Keychain Manager was not loaded. Available in Mac OS X v10.0 and later.
`errKCReadOnly`	-25292	Returned by the function `KCCopyItem` to indicate that the keychain file is read-only and cannot be edited. Available in Mac OS X v10.0 and later.
`errKCAuthFailed`	-25293	Returned by the function `KCUnlock` to indicate that the authentication failed (too many unsuccessful retries). Available in Mac OS X v10.0 and later.
`errKCNoSuchKeychain`	-25294	Returned by the functions `KCUnlock`, `KCSetDefaultKeychain`, `KCGetStatus`, and `KCGetIndKeychain` to indicate that the specified keychain was not found. Available in Mac OS X v10.0 and later.
`errKCInvalidKeychain`	-25295	Returned by the functions `KCUnlock`, `KCSetDefaultKeychain`, `KCGetStatus`, `KCGetKeychainName`, `KCChangeSettings`, and `KCCreateKeychain` to indicate that the keychain is not valid. Available in Mac OS X v10.0 and later.
`errKCDuplicateKeychain`	-25296	Returned by the function `KCCreateKeychain` to indicate that your application tried to create a keychain that already exists. Available in Mac OS X v10.0 and later.
`errKCDuplicateCallback`	-25297	Returned by the function `KCAddCallback` to indicate that your callback function was already registered. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errKCInvalidCallback	-25298	Returned by the function KCRemoveCallback to indicate that the callback function was not previously registered. Available in Mac OS X v10.0 and later.
errKCDuplicateItem	-25299	Returned by the functions KCAddAppleSharePassword, KCAddInternetPassword, KCAddInternetPasswordWithPath, KCAddGenericPassword, and KCAddItem to indicate that you tried to add an existing keychain item to the keychain. Available in Mac OS X v10.0 and later.
errKCItemNotFound	-25300	Returned by the functions KCFindAppleSharePassword, KCFindInternetPassword, KCFindInternetPasswordWithPath, KCFindGenericPassword, KCFindNextItem, and KCFindFirstItem to indicate that no matching item was found. Available in Mac OS X v10.0 and later.
errKCBufferTooSmall	-25301	Returned by the functions KCFindAppleSharePassword, KCFindInternetPassword, KCFindInternetPasswordWithPath, KCFindGenericPassword, KCGetAttribute, KCGetData, and KCFindX509Certificates to indicate that the buffer was not large enough to contain the password data. Available in Mac OS X v10.0 and later.
errKCDataTooLarge	-25302	Returned by the functions KCAddAppleSharePassword, KCAddInternetPassword, KCAddInternetPasswordWithPath, KCAddGenericPassword, KCSetAttribute, and KCSetData to indicate that the data is too large. Available in Mac OS X v10.0 and later.
errKCNoSuchAttr	-25303	Returned by the functions KCSetAttribute, KCGetAttribute, and KCFindFirstItem to indicate that no such attribute exists. Available in Mac OS X v10.0 and later.
errKCInvalidItemRef	-25304	Returned by the functions KCSetAttribute, KCGetAttribute, KCSetData, KCGetData, KCAddItem, KCDeleteItem, KCUpdateItem, KCCopyItem, and KCGetKeychain to indicate that the keychain item reference is invalid. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errKCInvalidSearchRef	-25305	Returned by the functions KCFindNextItem and KCReleaseSearch to indicate that the specified search reference is invalid. Available in Mac OS X v10.0 and later.
errKCNoSuchClass	-25306	Returned by the function KCCopyItem to indicate that the item class does not exist. Available in Mac OS X v10.0 and later.
errKCNoDefaultKeychain	-25307	Returned by the functions KCChangeSettings, KCSetDefaultKeychain, KCGetDefaultKeychain, KCAddAppleSharePassword, KCAddInternetPassword, KCAddInternetPasswordWithPath, KCAddGenericPassword, KCFindAppleSharePassword, KCFindInternetPassword, KCFindInternetPasswordWithPath, KCFindGenericPassword, KCCopyItem, KCAddItem, KCDeleteItem, KCUpdateItem, KCFindNextItem, KCFindFirstItem, and KCFindX509Certificates to indicate that there is no default keychain. Available in Mac OS X v10.0 and later.
errKCInteractionNotAllowed	-25308	Returned by the functions KCCreateKeychain, KCChangeSettings, KCUnlock, and KCGetData (the latter two only when the Unlock Dialog and Allow Access dialog boxes are needed) to indicate that there is no start-up keychain. Available in Mac OS X v10.0 and later.
errKCReadOnlyAttr	-25309	Returned by the function KCSetAttribute to indicate that the keychain item attribute is read-only. Available in Mac OS X v10.0 and later.
errKCWrongKCVersion	-25310	Indicates that the wrong version of Keychain Manager is installed to perform this operation. Available in Mac OS X v10.0 and later.
errKCKeySizeNotAllowed	-25311	Indicates that the key size is illegal. Available in Mac OS X v10.0 and later.
errKCNoStorageModule	-25312	Returned by functions that prompts the loading of the Keychain Manager to indicate that the storage module is not found. Available in Mac OS X v10.0 and later.
errKCNoCertificateModule	-25313	Returned when a function is required for a certificate and the certificate module is not found. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`errKCNoPolicyModule`	-25314	Returned when a function is required for a trust policy and the policy module is not found. Available in Mac OS X v10.0 and later.
`errKCInteractionRequired`	-25315	Returned by the function `KCUnlock` to indicate that user interaction is required for this operation. Available in Mac OS X v10.0 and later.
`errKCDataNotAvailable`	-25316	Indicates that the requested data is not available. Available in Mac OS X v10.0 and later.
`errKCDataNotModifiable`	-25317	Returned by the functions `KCSetData` and `KCGetData` to indicate that the data cannot be modified. Available in Mac OS X v10.0 and later.
`errKCCreateChainFailed`	-25318	Returned by the functions `KCChooseCertificate` and `KCFindX509Certificates` to indicate that the attempt to create a new keychain failed. Available in Mac OS X v10.0 and later.

Launch Services Reference

Framework:	ApplicationServices/ApplicationServices.h
Declared in	LSInfo.h LSOpen.h
Companion guide	Launch Services Programming Guide

Overview

Mac OS X Launch Services is an API that enables a running application to open other applications or their document files in a way similar to the Finder or the Dock. Using Launch Services, an application can perform such tasks as:

- Open (**launch** or activate) another application
- Open a document or a URL (uniform resource locator) in another application
- Identify the preferred application for opening a given document or URL
- Register information about the kinds of document files and URLs an application is capable of opening
- Obtain appropriate information for displaying a file or URL on the screen, such as its icon, display name, and kind string
- Maintain and update the contents of the Recent Items menu

Although most of these services are normally performed by the Finder, other applications may also find them useful for purposes such as opening email attachments, following URLs embedded in a document, running helper applications, or opening embedded document components that were created by another application or require it for viewing or editing.

Many of Launch Services' capabilities were formerly provided by the Desktop Manager. With the advent of Mac OS X application bundles, however, the Desktop Manager has lost its usefulness, since it is not knowledgeable about bundled applications and simply ignores them. Similarly, Launch Services' facilities for dealing with URLs were formerly implemented through the Internet Config API. Launch Services replaces and supersedes the Desktop Manager and Internet Config with a new API providing similar functionality, but designed to operate properly in the Mac OS X environment.

Launch Services was created specifically to avoid the common need for applications to ask the Finder to open an application, document, or URL for them. In the past, opening such items in a way similar to the Finder required knowledge of several APIs, including the Desktop Manager, File Manager, Translation Manager, Internet Config, Process Manager, and Apple Event Manager. The Finder also had implicit knowledge of the desktop database and other information not available elsewhere for determining the correct application with which to open a given document.

Launch Services removes this specialized knowledge from the Finder and isolates it in a single, straightforward API available to any application. The Mac OS X Finder itself uses Launch Services to open applications, documents, and URLs at the user's request. Since the Finder does no additional processing beyond calling Launch Services, any client using Launch Services for these purposes is guaranteed to behave identically to the Finder itself.

Before reading this document, you should be familiar with the related document, *Launch Services Programming Guide*, which presents a conceptual overview of Launch Services and its operations.

Functions by Task

This section describes the functions defined in the Launch Services API.

Locating an Application

The functions described in this section locate the preferred application for opening a given item or family of items or the application matching a given set of defining characteristics, or test whether an application can open a designated item.

`LSGetApplicationForItem` (page 1219)
: Locates the preferred application for opening an item designated by file-system reference.

`LSGetApplicationForURL` (page 1220)
: Locates the preferred application for opening an item designated by URL.

`LSGetApplicationForInfo` (page 1218)
: Locates the preferred application for opening items with a specified file type, creator signature, filename extension, or any combination of these characteristics.

`LSCopyApplicationForMIMEType` (page 1206)
: Locates the preferred application for opening items with a specified MIME type.

`LSCopyApplicationURLsForURL` (page 1207)
: Locates all known applications suitable for opening an item designated by URL.

`LSCanRefAcceptItem` (page 1203)
: Tests whether an application can accept (open) an item designated by file-system reference.

`LSCanURLAcceptURL` (page 1204)
: Tests whether an application can accept (open) an item designated by URL.

`LSFindApplicationForInfo` (page 1217)
: Locates an application with a specified creator signature, bundle ID, filename, or any combination of these characteristics.

Opening Items

The functions described in this section open a designated item or collection of items, or launch or activate a designated application.

`LSOpenApplication` (page 1223)
: Launches the specified application.

`LSOpenItemsWithRole` (page 1229)
Opens items specified as an array of values of type `FSRef` with a specified role.

`LSOpenURLsWithRole` (page 1230)
Opens one or more URLs with the specified roles.

`LSOpenFSRef` (page 1228)
Opens an item designated by file-system reference, in the default manner in its preferred application.

`LSOpenFromRefSpec` (page 1225)
Opens one or more items designated by file-system reference, in either their preferred applications or a designated application.

`LSOpenCFURLRef` (page 1224)
Opens an item designated by URL, in the default manner in its preferred application.

`LSOpenFromURLSpec` (page 1226)
Opens one or more items designated by URL, in either their preferred applications or a designated application.

Obtaining Information About an Item

The functions described in this section obtain requested information about an item.

`LSCopyItemInfoForRef` (page 1212)
Obtains requested information about an item designated by file-system reference.

`LSCopyItemInfoForURL` (page 1213)
Obtains requested information about an item designated by URL.

`LSCopyDisplayNameForRef` (page 1209)
Obtains the display name for an item designated by file-system reference.

`LSCopyDisplayNameForURL` (page 1210)
Obtains the display name for an item designated by URL.

`LSCopyKindStringForRef` (page 1214)
Obtains the kind string for an item designated by file-system reference.

`LSCopyKindStringForURL` (page 1216)
Obtains the kind string for an item designated by URL.

`LSCopyKindStringForTypeInfo` (page 1215)
Obtains a kind string for items with a specified file type, creator signature, filename extension, or any combination of these characteristics.

`LSCopyKindStringForMIMEType` (page 1213)
Obtains the kind string for a specified MIME type.

`LSCopyItemAttribute` (page 1210)
Obtains the value of an item's attribute.

`LSCopyItemAttributes` (page 1211)
Obtains multiple item attribute values as a dictionary.

Getting and Setting Filename Extension Information

The functions described in this section obtain information about an item's filename extension, or control whether the extension should be hidden or shown on the screen.

`LSGetExtensionInfo` (page 1221)
Obtains the starting index of the extension within a filename.

`LSSetExtensionHiddenForRef` (page 1234)
Specifies whether the filename extension for an item designated by file-system reference should be hidden or shown.

`LSSetExtensionHiddenForURL` (page 1235)
Specifies whether the filename extension for an item designated by URL should be hidden or shown.

Registering an Application

The functions described in this section register an application in the Launch Services database.

`LSRegisterFSRef` (page 1231)
Registers an application, designated by file-system reference, in the Launch Services database.

`LSRegisterURL` (page 1232)
Registers an application, designated by URL, in the Launch Services database.

Working With Role Handlers

The functions described in this section get and set application bundle identifiers for handlers of specified content types and URL schemes.

`LSCopyAllRoleHandlersForContentType` (page 1205)
Returns an array of application bundle identifiers for applications capable of handling a specified content type with the specified roles.

`LSCopyDefaultRoleHandlerForContentType` (page 1208)
Returns the application bundle identifier of the user's preferred default handler for the specified content type with the specified role.

`LSSetDefaultRoleHandlerForContentType` (page 1233)
Sets the user's preferred default handler for the specified content type in the specified roles.

`LSGetHandlerOptionsForContentType` (page 1222)
Gets the handler options for the specified content type.

`LSSetHandlerOptionsForContentType` (page 1235)
Sets the handler option for the specified content type.

`LSCopyAllHandlersForURLScheme` (page 1205)
Returns an array of application bundle identifiers for applications capable of handling the specified URL scheme.

`LSCopyDefaultHandlerForURLScheme` (page 1207)
Returns the application bundle identifier of the user's preferred default handler for the specified URL scheme.

`LSSetDefaultHandlerForURLScheme` (page 1232)
Sets the user's preferred default handler for the specified URL scheme.

Functions No Longer Used

The functions described in this section are no longer used.

`LSInit` (page 1222) Deprecated in Mac OS X v10.3
(Deprecated. Formerly used to initialize Launch Services; now does nothing.)

`LSTerm` (page 1236) Deprecated in Mac OS X v10.3
(Deprecated. Formerly used to terminate Launch Services; now does nothing.)

Functions

LSCanRefAcceptItem

Tests whether an application can accept (open) an item designated by file-system reference.

```
OSStatus LSCanRefAcceptItem (
   const FSRef *inItemFSRef,
   const FSRef *inTargetRef,
   LSRolesMask inRoleMask,
   LSAcceptanceFlags inFlags,
   Boolean *outAcceptsItem
);
```

Parameters

inItemFSRef
A pointer to a file-system reference designating the source item (the item to test for acceptance by the target application); see the *File Manager Reference* in the Carbon File Management Documentation for a description of the `FSRef` data type.

inTargetFSRef
A pointer to a file-system reference designating the target application; see the *File Manager Reference* in the Carbon File Management Documentation for a description of the `FSRef` data type.

inRolesMask
A bit mask specifying the target application's desired role or roles with respect to the source item; see "Roles Mask" (page 1241) for a description of this mask. If the role is unimportant, pass `kLSRolesAll`.

inFlags
Flags specifying behavior to observe during the acceptance test; see "Acceptance Flags" (page 1248) for a description of these flags.

outAcceptsItem
A pointer to a Boolean value that, on return, will indicate whether the target application can accept the source item with at least one of the specified roles.

Return Value

A result code; see "Launch Services Result Codes" (page 1251).

Version Notes

Thread-safe since Mac OS version 10.2.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTCarbonShell

Declared In
`LSInfo.h`

LSCanURLAcceptURL

Tests whether an application can accept (open) an item designated by URL.

```
OSStatus LSCanURLAcceptURL (
   CFURLRef inItemURL,
   CFURLRef inTargetURL,
   LSRolesMask inRoleMask,
   LSAcceptanceFlags inFlags,
   Boolean *outAcceptsItem
);
```

Parameters

inItemURL

A Core Foundation URL reference designating the source item (the item to test for acceptance by the target application); see the *CFURL Reference* in the Core Foundation Reference Documentation for a description of the `CFURLRef` data type.

inTargetURL

A Core Foundation URL reference designating the target application; see the *CFURL Reference* in the Core Foundation Reference Documentation for a description of the `CFURLRef` data type. The URL must have scheme `file` and contain a valid path to an application file or application bundle.

inRolesMask

A bit mask specifying the target application's desired role or roles with respect to the source item; see "Roles Mask" (page 1241) for a description of this mask. This parameter applies only to URLs with a scheme component of `file`, and is ignored for all other schemes. If the role is unimportant, pass `kLSRolesAll`.

inFlags

Flags specifying behavior to observe during the acceptance test; see "Acceptance Flags" (page 1248) for a description of these flags.

outAcceptsItem

A pointer to a Boolean value that, on return, will indicate whether the target application can accept the source item with at least one of the specified roles.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
If the item URL's scheme is `file` (designating either a file or a directory), the acceptance test is based on the designated item's filename extension, file type, and creator signature, along with the role specified by the `inRolesMask` parameter; otherwise, it is based on the URL scheme (such as `http`, `ftp`, or `mailto`).

Version Notes
Thread-safe since Mac OS version 10.2.

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

Declared In
`LSInfo.h`

LSCopyAllHandlersForURLScheme

Returns an array of application bundle identifiers for applications capable of handling the specified URL scheme.

```
CFArrayRef LSCopyAllHandlersForURLScheme (
   CFStringRef inURLScheme
);
```

Parameters

inURLScheme

The URL scheme for which the application bundle identifiers are to be returned.

Return Value

An array containing the application bundle identifiers for applications capable of handling the URL scheme specified by `inURLScheme`, or `NULL` if no handlers are available.

Discussion

This function returns all of the application bundle identifiers that are capable of handling the specified URL scheme.

URL handling capability is determined according to the value of the `CFBundleURLTypes` key in an application's `Info.plist`. For information on the `CFBundleURLTypes` key, see the section "CFBundleURLTypes" in *Mac OS X Runtime Configuration Guidelines*.

Version Notes

Thread-safe since Mac OS X v10.4.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`LSInfo.h`

LSCopyAllRoleHandlersForContentType

Returns an array of application bundle identifiers for applications capable of handling a specified content type with the specified roles.

```
CFArrayRef LSCopyAllRoleHandlersForContentType (
   CFStringRef inContentType,
   LSRolesMask inRole
);
```

Parameters

inContentType

The content type. The content type is a uniform type identifier (UTI).

inRole

The role. Pass `kLSRolesAll` if any role is acceptable. For additional possible values, see "Roles Mask" (page 1241).

Return Value

The application bundle identifiers for applications capable of handling the specified content type in the specified roles, or `NULL` if no handlers are available.

Discussion
This function returns all of the application bundle identifiers that are capable of handling the specified content type in the specified roles.

The `CFBundleDocumentTypes` key in an application's `Info.plist` can be used to set an application's content handling capabilities. The `LSItemContentTypes` key is particularly useful because it supports the use of UTIs in document claims. For information on the `CFBundleDocumentTypes` key, see the section "CFBundleDocumentTypes" in *Mac OS X Runtime Configuration Guidelines*.

Version Notes
Thread-safe since Mac OS X v10.4.

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

Declared In
`LSInfo.h`

LSCopyApplicationForMIMEType

Locates the preferred application for opening items with a specified MIME type.

```
OSStatus LSCopyApplicationForMIMEType (
   CFStringRef inMIMEType,
   LSRolesMask inRoleMask,
   CFURLRef *outAppURL
);
```

Parameters

inMIMEType
A Core Foundation string object specifying the MIME type to consider; see the *CFString Reference* in the Core Foundation Reference Documentation for a description of the `CFStringRef` data type. Comparison of MIME types is case-insensitive.

inRolesMask
A bit mask specifying the application's desired role or roles with respect to items with the specified MIME type; see "Roles Mask" (page 1241) for a description of this mask. If the role is unimportant, pass `kLSRolesAll`.

outAppURL
A pointer to a Core Foundation URL reference that, on return, will identify the preferred application for items with the specified MIME type; see the *CFURL Reference* in the Core Foundation Reference Documentation for a description of the `CFURLRef` data type. You are responsible for releasing the URL reference object.

Return Value
A result code; see "Launch Services Result Codes" (page 1251). If no application suitable for opening items with the specified MIME type is found in the Launch Services database, the function will return the result code `kLSApplicationNotFoundErr`.

Version Notes
Thread-safe since Mac OS version 10.2.

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

Declared In
`LSInfo.h`

LSCopyApplicationURLsForURL

Locates all known applications suitable for opening an item designated by URL.

```
CFArrayRef LSCopyApplicationURLsForURL (
   CFURLRef inURL,
   LSRolesMask inRoleMask
);
```

Parameters

inURL

A Core Foundation URL reference designating the item for which all suitable applications are requested; see the *CFURL Reference* in the Core Foundation Reference Documentation for a description of the `CFURLRef` data type.

inRolesMask

A bit mask specifying the applications' desired role or roles with respect to the designated item; see "Roles Mask" (page 1241) for a description of this mask. This parameter applies only to URLs with a scheme component of `file`, and is ignored for all other schemes. If the role is unimportant, pass `kLSRolesAll`.

Return Value
An array of Core Foundation URL references, one for each application that can open the designated item with at least one of the specified roles. You are responsible for releasing the array object. If no suitable applications are found in the Launch Services database, the function will return `NULL`

Discussion
If the item URL's scheme is `file` (designating either a file or a directory), the selection of suitable applications is based on the designated item's filename extension, file type, and creator signature, along with the role specified by the `inRolesMask` parameter; otherwise, it is based on the URL scheme (such as `http`, `ftp`, or `mailto`).

Version Notes
Thread-safe since Mac OS version 10.3.

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

Declared In
`LSInfo.h`

LSCopyDefaultHandlerForURLScheme

Returns the application bundle identifier of the user's preferred default handler for the specified URL scheme.

```
CFStringRef LSCopyDefaultHandlerForURLScheme (
   CFStringRef inURLScheme
);
```

Parameters

inURLScheme

The URL scheme for which the application bundle identifier is to be returned.

Return Value

The application bundle identifier of the specified URL scheme.

Discussion

This function returns the user's currently preferred default handler for the specified URL scheme.

URL handling capability is determined according to the value of the `CFBundleURLTypes` key in an application's `Info.plist`. For information on the `CFBundleURLTypes` key, see the section "CFBundleURLTypes" in *Mac OS X Runtime Configuration Guidelines*.

Version Notes

Thread-safe since Mac OS X v10.4.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`LSInfo.h`

LSCopyDefaultRoleHandlerForContentType

Returns the application bundle identifier of the user's preferred default handler for the specified content type with the specified role.

```
CFStringRef LSCopyDefaultRoleHandlerForContentType (
   CFStringRef inContentType,
   LSRolesMask inRole
);
```

Parameters

inContentType

The content type. The content type is a uniform type identifier (UTI).

inRole

The role. Pass `kLSRolesAll` if any role is acceptable. For additional possible values, see "Roles Mask" (page 1241).

Return Value

The application bundle identifier of the default handler for the specified content type in the specified roles, or `NULL` if no handler is available.

Discussion

This function returns the user's currently preferred default handler for the specified content type. Say, for example, that `LSSetDefaultRoleHandlerForContentType` (page 1233) has been used to set "com.Apple.TextEdit" for the "public.xml" content type. When a file whose content type is "public.xml" is double-clicked, TextEdit will be launched to open the file. If you call `LSCopyDefaultRoleHandlerForContentType (CFSTR("public.xml"), kLSRolesAll)`, the string `com.apple.TextEdit` is returned.

The `CFBundleDocumentTypes` key in an application's `Info.plist` can be used to set an application's content handling capabilities. The `LSItemContentTypes` key is particularly useful because it supports the use of UTIs in document claims. For information on the `CFBundleDocumentTypes` key, see the section "CFBundleDocumentTypes" in *Mac OS X Runtime Configuration Guidelines*.

Version Notes
Thread-safe since Mac OS X v10.4.

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

Declared In
`LSInfo.h`

LSCopyDisplayNameForRef

Obtains the display name for an item designated by file-system reference.

```
OSStatus LSCopyDisplayNameForRef (
   const FSRef *inRef,
   CFStringRef *outDisplayName
);
```

Parameters

`inRef`

A pointer to a file-system reference designating the item whose display name is requested; see the *File Manager Reference* in the Carbon File Management Documentation for a description of the `FSRef` data type.

`outDisplayName`

A pointer to a Core Foundation string object that, on return, will contain the item's display name; see the *CFString Reference* in the Core Foundation Reference Documentation for a description of the `CFStringRef` data type. You are responsible for releasing this object.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
The item's display name is returned in the form in which it will appear on the user's screen; it may be localized (for applications and folders), and it excludes the filename extension if the extension is set to be hidden and the Finder preference to always show extensions is not enabled.

Version Notes
Thread-safe since Mac OS version 10.2.

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

Related Sample Code
QTMetaData

Declared In
`LSInfo.h`

LSCopyDisplayNameForURL

Obtains the display name for an item designated by URL.

```
OSStatus LSCopyDisplayNameForURL (
   CFURLRef inURL,
   CFStringRef *outDisplayName
);
```

Parameters

inFileURL

A Core Foundation URL reference designating the item whose display name is requested; see the *CFURL Reference* in the Core Foundation Reference Documentation for a description of the `CFURLRef` data type. The URL must have scheme `file` and contain a valid path to either a file or a directory.

outDisplayName

A pointer to a Core Foundation string object that, on return, will contain the item's display name; see the *CFString Reference* in the Core Foundation Reference Documentation for a description of the `CFStringRef` data type. You are responsible for releasing this object.

Return Value

A result code; see "Launch Services Result Codes" (page 1251).

Discussion

The item's display name is returned in the form in which it will appear on the user's screen; it may be localized (for applications and folders), and it excludes the filename extension if the extension is set to be hidden and the Finder preference to always show extensions is not enabled.

Version Notes

Thread-safe since Mac OS version 10.2.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`LSInfo.h`

LSCopyItemAttribute

Obtains the value of an item's attribute.

```
OSStatus LSCopyItemAttribute (
   const FSRef *inItem,
   LSRolesMask inRoles,
   CFStringRef inAttributeName,
   CFTypeRef *outValue
);
```

Parameters

inItem

The `FSRef` of the item to query.

inRoles

The roles. When obtaining attributes related to document binding (such as `kLSItemRoleHandlerDisplayName`), at least one of the roles must be provided by the application selected. Pass `kLSRolesAll` if any role is acceptable.

inAttributeName

The name of the attribute to copy. For possible values, see "Item Attribute Constants" (page 1245).

outValue

A pointer to a `CFTypeRef`. On return, the `CFTypeRef` is set to the copied attribute value (a CF object), or is `NULL` if an error occurs. The type of the returned object varies depending on the attribute that is requested.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Version Notes
Thread-safe since Mac OS X v10.4.

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

Declared In
`LSInfo.h`

LSCopyItemAttributes

Obtains multiple item attribute values as a dictionary.

```
OSStatus LSCopyItemAttributes (
   const FSRef *inItem,
   LSRolesMask inRoles,
   CFArrayRef inAttributeNames,
   CFDictionaryRef *outValues
);
```

Parameters

inItem

The `FSRef` of the item to query.

inRoles

The roles. When obtaining attributes related to document binding (such as `kLSItemRoleHandlerDisplayName`), at least one of the roles must be provided by the application selected. Pass `kLSRolesAll` if any role is acceptable.

inAttributeNames

A `CFArrayRef` for an array containing the attribute names to copy. For possible values, see "Item Attribute Constants" (page 1245).

outValues

On return, a pointer a `CFDictionaryRef` for a dictionary whose keys are the attribute names specified by the `inAttributeNames` parameter and whose values are the attribute's values. The `CFTypeID` of each value in the dictionary varies by attribute. See "Item Attribute Constants" (page 1245) for the data type of each value. If the item does not have a specified attribute, the key for the attribute is not in the dictionary.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Version Notes
Thread-safe since Mac OS X v10.4.

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

Declared In
`LSInfo.h`

LSCopyItemInfoForRef

Obtains requested information about an item designated by file-system reference.

```
OSStatus LSCopyItemInfoForRef (
   const FSRef *inItemRef,
   LSRequestedInfo inWhichInfo,
   LSItemInfoRecord *outItemInfo
);
```

Parameters

inItemRef

A pointer to a file-system reference designating the item about which information is requested; see the *File Manager Reference* in the Carbon File Management Documentation for a description of the `FSRef` data type.

inWhichInfo

Flags specifying what information to obtain; see "Requested-Information Flags" (page 1244) for a description of these flags.

outItemInfo

A pointer to an item-information record that, on return, will contain the requested information; see LSItemInfoRecord (page 1240) for a description of this structure.

If you request the item's filename extension (field `extension` of the item-information record, requested by flag `kLSRequestExtension`), you are responsible for releasing the Core Foundation string object in which the extension is returned.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
The information obtained about an item can include its filename extension, file type, creator signature, and various item-information flags (indicating, for example, whether the item is an application, or whether it has a hidden extension); see "Item-Information Flags" (page 1246) for a description of these flags.

Version Notes
Thread-safe since Mac OS version 10.2.

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

Related Sample Code
QTCarbonShell

Declared In
`LSInfo.h`

LSCopyItemInfoForURL

Obtains requested information about an item designated by URL.

```
OSStatus LSCopyItemInfoForURL (
   CFURLRef inURL,
   LSRequestedInfo inWhichInfo,
   LSItemInfoRecord *outItemInfo
);
```

Parameters

inFileURL

A Core Foundation URL reference designating the item about which information is requested; see the *CFURL Reference* in the Core Foundation Reference Documentation for a description of the `CFURLRef` data type. The URL must have scheme `file` and contain a valid path to either a file or a directory.

inWhichInfo

Flags specifying what information to obtain; see "Requested-Information Flags" (page 1244) for a description of these flags.

outItemInfo

A pointer to an item-information record that, on return, will contain the requested information; see LSItemInfoRecord (page 1240) for a description of this structure.

If you request the item's filename extension (field `extension` of the item-information record, requested by flag `kLSRequestExtension`), you are responsible for releasing the Core Foundation string object in which the extension is returned.

Return Value

A result code; see "Launch Services Result Codes" (page 1251).

Discussion

The information obtained about an item can include its filename extension, file type, creator signature, and various item-information flags (indicating, for example, whether the item is an application, or whether it has a hidden extension); see "Item-Information Flags" (page 1246) for a description of these flags.

Version Notes

Thread-safe since Mac OS version 10.2.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonCocoa_PictureCursor

Declared In

`LSInfo.h`

LSCopyKindStringForMIMEType

Obtains the kind string for a specified MIME type.

```
OSStatus LSCopyKindStringForMIMEType (
   CFStringRef inMIMEType,
   CFStringRef *outKindString
);
```

Parameters

inMIMEType

A Core Foundation string object specifying the MIME type whose kind string is requested; see the *CFString Reference* in the Core Foundation Reference Documentation for a description of the `CFStringRef` data type. Comparison of MIME types is case-insensitive.

outKindString

A pointer to a Core Foundation string object that, on return, will contain the kind string for the specified MIME type; see the *CFString Reference* in the Core Foundation Reference Documentation for a description of the `CFStringRef` data type. You are responsible for releasing this object.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
The kind string (which may be localized) is obtained from the preferred application for opening items of the specified the MIME type, if one is found in the Launch Services database; otherwise, a more generic kind string is chosen.

Version Notes
Thread-safe since Mac OS version 10.2.

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

Declared In
`LSInfo.h`

LSCopyKindStringForRef

Obtains the kind string for an item designated by file-system reference.

```
OSStatus LSCopyKindStringForRef (
   const FSRef *inFSRef,
   CFStringRef *outKindString
);
```

Parameters

inFSRef

A pointer to a file-system reference designating the item whose kind string is requested; see the *File Manager Reference* in the Carbon File Management Documentation for a description of the `FSRef` data type.

outKindString

A pointer to a Core Foundation string object that, on return, will contain the item's kind string; see the *CFString Reference* in the Core Foundation Reference Documentation for a description of the `CFStringRef` data type. You are responsible for releasing this object.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
The kind string (which may be localized) is obtained from the item's preferred application, if one is found in the Launch Services database; otherwise, a more generic kind string is chosen. For example, the kind string might be `FrameMaker Document`, or just `Document` if the item is a document for which no application is found.

Version Notes
Thread-safe since Mac OS version 10.2.

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

Declared In
`LSInfo.h`

LSCopyKindStringForTypeInfo

Obtains a kind string for items with a specified file type, creator signature, filename extension, or any combination of these characteristics.

```
OSStatus LSCopyKindStringForTypeInfo (
   OSType inType,
   OSType inCreator,
   CFStringRef inExtension,
   CFStringRef *outKindString
);
```

Parameters

inType

The file type to consider. Comparison of file types is case-sensitive. Pass `kLSUnknownType` if the items' file type is unimportant.

inCreator

The creator signature to consider. Comparison of creator signatures is case-sensitive. Pass `kLSUnknownCreator` if the items' creator signature is unimportant.

inExtension

A Core Foundation string object specifying the filename extension to consider; see the *CFString Reference* in the Core Foundation Reference Documentation for a description of the `CFStringRef` data type. Comparison of filename extensions is case-insensitive. Pass `NULL` if the items' filename extension is unimportant.

outKindString

A pointer to a Core Foundation string object that, on return, will contain the requested kind string; see the *CFString Reference* in the Core Foundation Reference Documentation for a description of the `CFStringRef` data type. You are responsible for releasing this object.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
This function obtains the kind string that most closely describes items having the specified characteristics. It is useful when you want to display the kind string for a document you do not yet have (such as an email attachment).

You can request any combination of one, two, or all three of the characteristics specified by the *inType*, *inCreator*, and *inExtension* parameters; at least one of these characteristics must be supplied. The kind string (which may be localized) is obtained from the preferred application for opening such items, if one is found in the Launch Services database; otherwise, a more generic kind string is chosen. For example, the kind string might be `FrameMaker Document`, or just `Document` if no suitable application is found.

Note that since the choice of a preferred application is subject to any document binding preferences the user may have set, the kind string will not necessarily be obtained from the default application that matches the specified creator signature (if any), but may instead be taken from a user-specified application that overrides the default. For example, if the user has specified that files of type `'PDF '` and creator `'ACRO'` should be opened in the Preview application rather than in Acrobat, the kind string for this combination of characteristics will be that defined for `'PDF '` files by Preview and not by Acrobat.

Version Notes
Thread-safe since Mac OS version 10.2

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

Declared In
`LSInfo.h`

LSCopyKindStringForURL

Obtains the kind string for an item designated by URL.

```
OSStatus LSCopyKindStringForURL (
   CFURLRef inURL,
   CFStringRef *outKindString
);
```

Parameters

inURL

A Core Foundation URL reference designating the item whose kind string is requested; see the *CFURL Reference* in the Core Foundation Reference Documentation for a description of the `CFURLRef` data type.

outKindString

A pointer to a Core Foundation string object that, on return, will contain the item's kind string; see the *CFString Reference* in the Core Foundation Reference Documentation for a description of the `CFStringRef` data type. You are responsible for releasing this object.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
The kind string (which may be localized) is obtained from the item's preferred application, if one is found in the Launch Services database; otherwise, a more generic kind string is chosen. For example, the kind string might be `FrameMaker Document`, or just `Document` if the item is a document for which no application is found. If the item URL's scheme is `file` (designating either a file or a directory), the selection of the preferred application is based on the designated item's filename extension, file type, and creator signature; otherwise, it is based on the URL scheme (such as `http`, `ftp`, or `mailto`).

Version Notes
Thread-safe since Mac OS version 10.2.

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

Declared In
`LSInfo.h`

LSFindApplicationForInfo

Locates an application with a specified creator signature, bundle ID, filename, or any combination of these characteristics.

```
OSStatus LSFindApplicationForInfo (
   OSType inCreator,
   CFStringRef inBundleID,
   CFStringRef inName,
   FSRef *outAppRef,
   CFURLRef *outAppURL
);
```

Parameters

inCreator

The creator signature to consider. Comparison of creator signatures is case-sensitive. Pass `kLSUnknownCreator` if the application's creator signature is unimportant.

inBundleID

A Core Foundation string object specifying the bundle ID to consider; see the *CFString Reference* in the Core Foundation Reference Documentation for a description of the `CFStringRef` data type. Comparison of bundle IDs is case-insensitive. Pass `NULL` if the application's bundle ID is unimportant.

inName

A Core Foundation string object specifying the filename to consider; see the *CFString Reference* in the Core Foundation Reference Documentation for a description of the `CFStringRef` data type. The string must include any extension (such as `'.app'`) that is part of the filename. Comparison of filenames is case-insensitive. Pass `NULL` if the application's filename is unimportant.

outAppRef

A pointer to a file-system reference that, on return, will identify the requested application; see the *File Manager Reference* in the Carbon File Management Documentation for a description of the `FSRef` data type. Pass `NULL` if you are not interested in identifying the application in this form; however, this parameter and `outAppURL` cannot both be `NULL`.

outAppURL

A pointer to a Core Foundation URL reference that, on return, will identify the requested application; see the *CFURL Reference* in the Core Foundation Reference Documentation for a description of the `CFURLRef` data type. Pass `NULL` if you are not interested in identifying the application in this form; however, this parameter and `outAppRef` cannot both be `NULL`.

Despite the absence of the word `Copy` in its name, this function retains the URL reference object on your behalf; you are responsible for releasing this object.

Return Value
A result code; see "Launch Services Result Codes" (page 1251). If no suitable application is found in the Launch Services database, the function will return the result code `kLSApplicationNotFoundErr`.

Discussion
You can request any combination of one, two, or all three of the characteristics specified by the *inCreator*, *inBundleID*, and *inName* parameters; at least one of these characteristics must be supplied. If more than one application is found matching the specified characteristics, Launch Services chooses one in the same manner as when locating the preferred application for opening an item.

Version Notes
Thread-safe since Mac OS version 10.2.

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

Declared In
`LSInfo.h`

LSGetApplicationForInfo

Locates the preferred application for opening items with a specified file type, creator signature, filename extension, or any combination of these characteristics.

```
OSStatus LSGetApplicationForInfo (
   OSType inType,
   OSType inCreator,
   CFStringRef inExtension,
   LSRolesMask inRoleMask,
   FSRef *outAppRef,
   CFURLRef *outAppURL
);
```

Parameters

inType
The file type to consider. Comparison of file types is case-sensitive. Pass `kLSUnknownType` if the items' file type is unimportant.

inCreator
The creator signature to consider. Comparison of creator signatures is case-sensitive. Pass `kLSUnknownCreator` if the items' creator signature is unimportant.

inExtension
A Core Foundation string object specifying the filename extension to consider; see the *CFString Reference* in the Core Foundation Reference Documentation for a description of the `CFStringRef` data type. Comparison of filename extensions is case-insensitive. Pass `NULL` if the items' filename extension is unimportant.

inRolesMask
A bit mask specifying the application's desired role or roles with respect to items with the specified characteristics; see "Roles Mask" (page 1241) for a description of this mask. If the role is unimportant, pass `kLSRolesAll`.

outAppRef
A pointer to a file-system reference that, on return, will identify the preferred application for opening items with the specified characteristics; see the *File Manager Reference* in the Carbon File Management Documentation for a description of the `FSRef` data type. Pass `NULL` if you are not interested in identifying the preferred application in this form; however, this parameter and `outAppURL` cannot both be `NULL`.

outAppURL

A pointer to a Core Foundation URL reference that, on return, will identify the preferred application for items with the specified characteristics; see the *CFURL Reference* in the Core Foundation Reference Documentation for a description of the `CFURLRef` data type. Pass `NULL` if you are not interested in identifying the preferred application in this form; however, this parameter and `outAppRef` cannot both be `NULL`.

Despite the absence of the word `Copy` in its name, this function retains the URL reference object on your behalf; you are responsible for releasing this object.

Return Value
A result code; see "Launch Services Result Codes" (page 1251). If no application suitable for opening items with the specified characteristics is found in the Launch Services database, the function will return the result code `kLSApplicationNotFoundErr`.

Discussion
You can request any combination of one, two, or all three of the characteristics specified by the *inType*, *inCreator*, and *inExtension* parameters; at least one of these characteristics must be supplied. Note that since the choice of a preferred application is subject to any document binding preferences the user may have set, the application chosen will not necessarily be the default application that matches the input characteristics, but may instead be a user-specified application that overrides the default.

Version Notes
Thread-safe since Mac OS version 10.2.

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

Declared In
`LSInfo.h`

LSGetApplicationForItem

Locates the preferred application for opening an item designated by file-system reference.

```
OSStatus LSGetApplicationForItem (
   const FSRef *inItemRef,
   LSRolesMask inRoleMask,
   FSRef *outAppRef,
   CFURLRef *outAppURL
);
```

Parameters

inItemRef

A pointer to a file-system reference designating the item whose preferred application is requested; see the *File Manager Reference* in the Carbon File Management Documentation for a description of the `FSRef` data type.

inRolesMask

A bit mask specifying the application's desired role or roles with respect to the designated item; see "Roles Mask" (page 1241) for a description of this mask. If the role is unimportant, pass `kLSRolesAll`.

outAppRef

A pointer to a file-system reference that, on return, will identify the item's preferred application; see the *File Manager Reference* in the Carbon File Management Documentation for a description of the `FSRef` data type. Pass `NULL` if you are not interested in identifying the preferred application in this form; however, this parameter and `outAppURL` cannot both be `NULL`.

outAppURL

A pointer to a Core Foundation URL reference that, on return, will identify the item's preferred application; see the *CFURL Reference* in the Core Foundation Reference Documentation for a description of the `CFURLRef` data type. Pass `NULL` if you are not interested in identifying the preferred application in this form; however, this parameter and `outAppRef` cannot both be `NULL`.

Despite the absence of the word `Copy` in its name, this function retains the URL reference object on your behalf; you are responsible for releasing this object.

Return Value

A result code; see "Launch Services Result Codes" (page 1251). If no application suitable for opening the item is found in the Launch Services database, the function will return the result code `kLSApplicationNotFoundErr`.

Version Notes

Thread-safe since Mac OS version 10.2.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`LSInfo.h`

LSGetApplicationForURL

Locates the preferred application for opening an item designated by URL.

```
OSStatus LSGetApplicationForURL (
   CFURLRef inURL,
   LSRolesMask inRoleMask,
   FSRef *outAppRef,
   CFURLRef *outAppURL
);
```

Parameters

inURL

A Core Foundation URL reference designating the item whose preferred application is requested; see the *CFURL Reference* in the Core Foundation Reference Documentation for a description of the `CFURLRef` data type.

inRolesMask

A bit mask specifying the application's desired role or roles with respect to the designated item; see "Roles Mask" (page 1241) for a description of this mask. This parameter applies only to URLs with a scheme component of `file`, and is ignored for all other schemes. If the role is unimportant, pass `kLSRolesAll`.

outAppRef

A pointer to a file-system reference that, on return, will identify the item's preferred application; see the *File Manager Reference* in the Carbon File Management Documentation for a description of the `FSRef` data type. Pass `NULL` if you are not interested in identifying the preferred application in this form; however, this parameter and `outAppURL` cannot both be `NULL`.

outAppURL

A pointer to a Core Foundation URL reference that, on return, will identify the item's preferred application; see the *CFURL Reference* in the Core Foundation Reference Documentation for a description of the `CFURLRef` data type. Pass `NULL` if you are not interested in identifying the preferred application in this form; however, this parameter and `outAppRef` cannot both be `NULL`.

Despite the absence of the word `Copy` in its name, this function retains the URL reference object on your behalf; you are responsible for releasing this object.

Return Value
A result code; see "Launch Services Result Codes" (page 1251). If no application suitable for opening the item is found in the Launch Services database, the function will return the result code `kLSApplicationNotFoundErr`.

Discussion
If the item URL's scheme is `file` (designating either a file or a directory), the selection of the preferred application is based on the designated item's filename extension, file type, and creator signature, along with the role specified by the `inRolesMask` parameter; otherwise, it is based on the URL scheme (such as `http`, `ftp`, or `mailto`).

Version Notes
Thread-safe since Mac OS version 10.2.

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

Declared In
`LSInfo.h`

LSGetExtensionInfo

Obtains the starting index of the extension within a filename.

```
OSStatus LSGetExtensionInfo (
   UniCharCount inNameLen,
   const UniChar inNameBuffer[],
   UniCharCount *outExtStartIndex
);
```

Parameters
inNameLen

The number of characters in the filename specified by the *inNameBuffer* parameter.

inNameBuffer

The buffer containing the filename's Unicode characters.

outExtStartIndex

A pointer to a value of type `UniCharCount` that, on return, will give the starting index of the extension within the filename. If the name does not contain a valid extension (one with no spaces in it), the value on return will be `kLSInvalidExtensionIndex`.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
The starting index is the number of Unicode characters from the start of the filename buffer to the first character of the extension (not including the period).

Version Notes
Thread-safe since Mac OS version 10.2.

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

Related Sample Code
QTCarbonShell

Declared In
`LSInfo.h`

LSGetHandlerOptionsForContentType

Gets the handler options for the specified content type.

```
LSHandlerOptions LSGetHandlerOptionsForContentType (
   CFStringRef inContentType
);
```

Parameters

inContentType
The content type for which the handler options are to be obtained. The content type is a uniform type identifier (UTI).

Return Value
The handler option that is set for the specified content type. For possible values, see "Handler Option Constants" (page 1249).

Version Notes
Thread-safe since Mac OS X v10.4.

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

Declared In
`LSInfo.h`

LSInit

(Deprecated in Mac OS X v10.3. Formerly used to initialize Launch Services; now does nothing.)

Not recommended.

```
OSStatus LSInit (
   LSInitializeFlags inFlags
);
```

Discussion
Calling this function was formerly required in order to initialize Launch Services; it is no longer needed, however, because Launch Services is now initialized automatically the first time one of its functions is called. `LSInit` now does nothing at all.

Version Notes
Thread-safe since Mac OS version 10.2.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.

Declared In
`LSInfo.h`

LSOpenApplication

Launches the specified application.

```
OSStatus LSOpenApplication (
   const LSApplicationParameters *appParams,
   ProcessSerialNumber *outPSN
);
```

Parameters

`inAppParams`
A `LSApplicationParameters` (page 1237) structure specifying the application to launch and its launch parameters. This parameter cannot be `NULL`.

`outPSN`
On input, a pointer to a value of type `ProcessSerialNumber` that, on return, contains the process serial number (PSN) of the application specified by `inAppParams`, or `NULL` if you don't want to receive the PSN.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
The `LSOpenApplication` launches one application. This function is an updated alternative to the Process Manager's `LaunchApplication` function. Launch arguments are specified in the `inAppParams` argument, which must be supplied. If the application is already running in the current session, it is made the front process (unless the `kLSLaunchNewInstance` flag is used, which always causes a new process to be created).

If `outPSN` is not `NULL`, on return, the structure it points to contains the PSN of the launched (or activated) process. Note that for asynchronous launches, the application may not have finished launching when this function returns.

Version Notes
Thread-safe since Mac OS X v10.4.

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

Declared In
LSOpen.h

LSOpenCFURLRef

Opens an item designated by URL, in the default manner in its preferred application.

```
OSStatus LSOpenCFURLRef (
   CFURLRef inURL,
   CFURLRef *outLaunchedURL
);
```

Parameters

inURL

A Core Foundation URL reference designating the item to open; see the *CFURL Reference* in the Core Foundation Reference Documentation for a description of the `CFURLRef` data type.

outLaunchedURL

A pointer to a Core Foundation URL reference that, on return, will identify the application launched. Pass `NULL` if this information is unimportant.

Despite the absence of the word `Copy` in its name, this function retains the URL reference object on your behalf; you are responsible for releasing this object.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
The designated item is opened in the default manner, as if it had been opened with the `LSOpenFromURLSpec` function with a launch specification specifying the launch flag `kLSLaunchDefaults`: that is, asynchronously, starting the Classic emulation environment if necessary, and with the remaining launch parameters taken from the application's information property list. For greater control, call `LSOpenFromURLSpec` directly. See `"Launch Flags"` (page 1242) for more information about launch flags.

If the item URL's scheme is `file` (designating either a file or a directory), the selection of the preferred application is based on the designated item's filename extension, file type, and creator signature; otherwise, it is based on the URL scheme (such as `http`, `ftp`, or `mailto`). The application is launched or activated, as required, and sent an appropriate Apple event depending on the circumstances:

- If the URL's scheme is `file` and it designates a document, the document's preferred application is launched (or activated if it is already running).
  - If the application claims to accept `file` URLs, it is sent a `'GURL'` ("get URL") Apple event containing the item's URL.
  - If the application does not claim to accept `file` URLs, it is sent an `'odoc'` ("open document") Apple event identifying the document to open.
- If the URL's scheme is `file` and it designates an application:
  - If the application is not already running, it is launched and sent an `'oapp'` ("open application") Apple event.
  - If the application is already running, it is activated and sent an `'rapp'` ("reopen application") Apple event.

- If the URL has a scheme other than `file`, the scheme's preferred application is launched (or activated if it is already running) and sent a `'GURL'` ("get URL") Apple event containing the item's URL.

As of Mac OS X v10.4 and later, `LSOpenURLsWithRole` (page 1230) is the preferred way of opening a URL.

Version Notes
Thread-safe since Mac OS version 10.2.

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

Related Sample Code
BSDLLCTest
QTCarbonShell
SimpleDataQueue
SimpleUserClient

Declared In
`LSOpen.h`

LSOpenFromRefSpec

Opens one or more items designated by file-system reference, in either their preferred applications or a designated application.

```
OSStatus LSOpenFromRefSpec (
   const LSLaunchFSRefSpec *inLaunchSpec,
   FSRef *outLaunchedRef
);
```

Parameters

`inLaunchSpec`
A pointer to a file-based launch specification indicating what to open and how to launch the relevant application or applications; see LSLaunchFSRefSpec (page 1237) for a description of this structure.

`outLaunchedRef`
A pointer to a file-system reference that, on return, will identify the application launched; see the *File Manager Reference* in the Carbon File Management Documentation for a description of the `FSRef` data type. Pass `NULL` if this information is unimportant. If more than one application is launched, the one identified will be the one corresponding to the first item designated in the launch specification.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
This function affords greater control of how items are opened or applications launched than is possible with the `LSOpenFSRef` function. For instance, you can use it to open multiple items in a single call, in either the same or different applications; open documents for printing rather than for simple viewing or editing; or force a document to open in an application other than its own preferred application.

The launch specification supplied for the `inLaunchSpec` parameter may designate an application to launch, items to open, or both. The relevant application or applications are launched or activated, as required, and sent an appropriate Apple event depending on the circumstances:

- If the launch specification designates both items to open and an application with which to open them, the designated application is used to open all of the items. The application is launched (or activated if it is already running) and sent an `'odoc'` ("open document") Apple event containing the list of items to open; if the items are to be printed, the Apple event is `'pdoc'` ("print document") instead.

  Note: When both an application and a list of items are supplied, the designated application is asked to open all of the items, whether or not it claims the ability to do so. Launch Services does not report an error if the application is unable to open one or more of the items; any error processing is the application's responsibility.

- If the launch specification designates items to open but not an application with which to open them, each item is opened in its own preferred application. Each application is launched or activated and sent an `'odoc'` or `'pdoc'` Apple event, as described for the preceding case. (If two or more of the items have the same preferred application, the application receives a single `'odoc'` or `'pdoc'` event listing all of the relevant items.)
- If the launch specification designates only an application to launch (or if one or more of the items to open are applications):
  - If the application is not already running, it is launched and sent an `'oapp'` ("open application") Apple event.
  - If the application is already running, it is activated and sent an `'rapp'` ("reopen application") Apple event.

As of Mac OS X v10.4 and later, `LSOpenItemsWithRole` (page 1229) is the preferred way of opening items.

Version Notes
Thread-safe since Mac OS version 10.2.

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

Declared In
`LSOpen.h`

LSOpenFromURLSpec

Opens one or more items designated by URL, in either their preferred applications or a designated application.

```
OSStatus LSOpenFromURLSpec (
   const LSLaunchURLSpec *inLaunchSpec,
   CFURLRef *outLaunchedURL
);
```

Parameters

inLaunchSpec

A pointer to a URL-based launch specification indicating what to open and how to launch the relevant application or applications; see LSLaunchURLSpec (page 1238) for a description of this structure.

outLaunchedURL

A pointer to a Core Foundation URL reference that, on return, will identify the application launched; see the *CFURL Reference* in the Core Foundation Reference Documentation for a description of the `CFURLRef` data type. Pass `NULL` if this information is unimportant. If more than one application is launched, the one identified will be the one corresponding to the first item designated in the launch specification.

Despite the absence of the word `Copy` in its name, this function retains the URL reference object on your behalf; you are responsible for releasing this object.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
This function affords greater control of how items are opened or applications launched than is possible with the `LSOpenCFURLRef` function. For instance, you can use it to open multiple items in a single call, in either the same or different applications; open documents for printing rather than for simple viewing or editing; or force a document to open in an application other than its own preferred application.

The launch specification supplied for the `inLaunchSpec` parameter may designate an application to launch, items to open, or both. The relevant application or applications are launched or activated, as required, and sent an appropriate Apple event depending on the circumstances:

- If the launch specification designates both items to open and an application with which to open them, the designated application is used to open all of the items. The application is launched (or activated if it is already running) and sent one or more Apple events:
  - If one or more of the item URLs have scheme `file` and designate documents to open, and if the application claims to accept `file` URLs, it is sent a `'GURL'` ("get URL") Apple event for each such URL.
  - If one or more of the item URLs have scheme `file` and designate documents to open, and if the application does not claim to accept `file` URLs, it is sent a single `'odoc'` ("open document") Apple event containing the list of items to open; if the items are to be printed, the Apple event is `'pdoc'` ("print document") instead.
  - For each item URL with a scheme other than `file`, the application is sent a `'GURL'` ("get URL") Apple event containing the item's URL.

Note: When both an application and a list of items are supplied, the designated application is asked to open all of the items, whether or not it claims the ability to do so. Launch Services does not report an error if the application is unable to open one or more of the items; any error processing is the application's responsibility.

- If the launch specification designates items to open but not an application with which to open them, each item is opened in its own preferred application. Each application is launched or activated and sent one or more Apple events, as described for the preceding case. (If two or more of the items have the same preferred application, the application receives a single `'odoc'` or `'pdoc'` event listing all of the relevant items.)
- If the launch specification designates only an application to launch (or if one or more of the items to open are `file` URLs designating applications):
  - If the application is not already running, it is launched and sent an `'oapp'` ("open application") Apple event.

- If the application is already running, it is activated and sent an `'rapp'` (“reopen application”) Apple event.

As of Mac OS X v10.4 and later, `LSOpenURLsWithRole` (page 1230) is the preferred way of opening URLs.

Version Notes
Thread-safe since Mac OS version 10.2.

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

Declared In
`LSOpen.h`

LSOpenFSRef

Opens an item designated by file-system reference, in the default manner in its preferred application.

```
OSStatus LSOpenFSRef (
   const FSRef *inRef,
   FSRef *outLaunchedRef
);
```

Parameters

inRef
A pointer to a file-system reference designating the item to open; see the *File Manager Reference* in the Carbon File Management Documentation for a description of the `FSRef` data type.

outLaunchedRef
A pointer to a file-system reference that, on return, will identify the application launched. Pass `NULL` if this information is unimportant.

Return Value
A result code; see “Launch Services Result Codes” (page 1251).

Discussion
The designated item is opened in the default manner, as if it had been opened with the `LSOpenFromRefSpec` function with a launch specification specifying the launch flag `kLSLaunchDefaults`: that is, asynchronously, starting the Classic emulation environment if necessary, and with the remaining launch parameters taken from the application’s information property list. For greater control, call `LSOpenFromRefSpec` directly. See `“Launch Flags”` (page 1242) for more information about launch flags.

The application is launched or activated, as required, and sent an appropriate Apple event depending on the circumstances:

- If the item is a document, its preferred application is launched (or activated if it is already running) and sent an `'odoc'` (“open document”) Apple event.
- If the item is an application that is not already running, it is launched and sent an `'oapp'` (“open application”) Apple event.
- If the item is an application that is already running, it is activated and sent an `'rapp'` (“reopen application”) Apple event.

As of Mac OS X v10.4 and later, `LSOpenItemsWithRole` (page 1229) is the preferred way of opening an item.

Version Notes
Thread-safe since Mac OS version 10.2.

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

Declared In
`LSOpen.h`

LSOpenItemsWithRole

Opens items specified as an array of values of type `FSRef` with a specified role.

```
OSStatus LSOpenItemsWithRole (
   const FSRef *inItems,
   CFIndex inItemCount,
   LSRolesMask inRole,
   const AEKeyDesc *inAEParam,
   const LSApplicationParameters *inAppParams,
   ProcessSerialNumber *outPSNs,
   CFIndex inMaxPSNCount
);
```

Parameters

inItems
An array of values of type `FSRef`.

inItemCount
The number of items specified in `inItems`.

inRole
A value of type `LSRolesMask` specifying one or more roles. If the role doesn't matter, use `kLSRolesAll`. For possible values, see "Roles Mask" (page 1241). If the `inAppParams` parameter is not `NULL`, this parameter is ignored.

inAEParam
An `AEKeyDesc` that is to be attached to the Apple Event(s) generated by Launch Services with the specified `AEKeyword`. This parameter can be `NULL`.

inAppParams
An `LSApplicationParameters` (page 1237) structure specifying the application to launch and its launch parameters, in which case the `inRole` parameter is ignored. This parameter can be `NULL`, in which case an application is selected that can handle each input item in at least one of the roles specified by the `inRole` parameter.

outPSNs
On input, a pointer to a caller-allocated buffer or `NULL` if you don't want to receive process serial number (PSN) information. If not `NULL` on input, on return, the buffer contains at each index the PSN that was used to open the item at the same index of the input item array (`inItems`).

inMaxPSNCount
The maximum number of PSNs that the buffer pointed to by `outPSNs` can hold.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
This function opens the specified items with the specified role. You can optionally specify the application and launch parameters in the `inAppParams` parameter. If an application is specified in the `inAppParams` parameter, the `inRole` parameter is ignored and the application is launched (if necessary).

Each application (regardless of whether it is launched or already running) receives an `'odoc'` Apple Event specifying the items that are to be opened.

If the `inItems` array contains any applications, this function launches them only if the `kLSRolesShell` bit is set in the `inRoles` parameter to indicate that the operating system should use the application itself as the execution shell of the new process.

If not `NULL`, the `outPSNs` buffer is filled with the PSN that was used to open each item at the same index of the `inItems` array. The PSN capacity of the output buffer is specified by `inMaxPSNCount`.

Version Notes
Thread-safe since Mac OS X v10.4.

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

Declared In
`LSOpen.h`

LSOpenURLsWithRole

Opens one or more URLs with the specified roles.

```
OSStatus LSOpenURLsWithRole (
   CFArrayRef inURLs,
   LSRolesMask inRole,
   const AEKeyDesc *inAEParam,
   const LSApplicationParameters *inAppParams,
   ProcessSerialNumber *outPSNs,
   CFIndex inMaxPSNCount
);
```

Parameters

inURLs
An array of values of type `CFURLRef`.

inRole
A value of type `LSRolesMask` specifying one or more roles. If the role doesn't matter, use `kLSRolesAll`. For possible values, see "Roles Mask" (page 1241). This parameter is ignored if the `inAppParams` parameter is not `NULL`.

inAEParam
A value of type `AEKeyDesc` that is to be attached to the Apple Event(s) generated by Launch Services with the specified `AEKeyword`. This parameter can be `NULL`.

inAppParams
An `LSApplicationParameters` (page 1237) structure specifying the application to launch and its launch parameters, in which case the `inRole` parameter is ignored. This parameter can be `NULL`, in which case an application is selected that can handle each input URL in at least one of the roles specified by the `inRole` parameter.

outPSNs

On input, a pointer to a caller-allocated buffer or `NULL` if you don't want to receive process serial number (PSN) information. If not `NULL` on input, on return, the buffer contains at each index the PSN that was used to open the URL at the same index of the input URL array (`inURLs`).

inMaxPSNCount

The maximum number of PSNs that the buffer pointed to by `outPSNs` can hold.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
This function opens the URLs specified by `inURLs` with the roles specified by `inRole`.

Each launched application receives one or more `'GURL'` Apple Events specifying the URLs to be opened. You can also pass file URLs, which are interpreted as file system items and opened in the manner of `LSOpenItemsWithRole` (page 1229) (that is, a handler is selected based on the item's filesystem metadata).If `inURLs` contains any application URLs, this function launches them only if the `kLSRolesShell` bit is set in the `inRoles` parameter, in which case the application is its own shell.If not `NULL`, the `outPSNs` buffer is filled with the process serial numbers that were used to open each URL at the same index of the input URL array specified by the `inURLs` parameter. The PSN capacity of the output buffer is specified by `inMaxPSNCount`.

Version Notes
Thread-safe since Mac OS X v10.4.

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

Declared In
`LSOpen.h`

LSRegisterFSRef

Registers an application, designated by file-system reference, in the Launch Services database.

```
OSStatus LSRegisterFSRef (
   const FSRef *inRef,
   Boolean inUpdate
);
```

Parameters

inRef

A pointer to a file-system reference designating the application to be registered; see the *File Manager Reference* in the Carbon File Management Documentation for a description of the `FSRef` data type.

inUpdate

A Boolean value specifying whether Launch Services should update existing information registered for the application, if any. If this parameter is `false`, the application will not be registered if it has already been registered previously and its current modification date has not changed from when it was last registered; if the parameter is `true`, the application's registered information will be updated even if its modification date has not changed.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
This function adds the designated application and its document and URL claims (if any) to the Launch Services database, making the application a candidate for document and URL binding.

Version Notes
Thread-safe since Mac OS version 10.3.

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

Declared In
`LSInfo.h`

LSRegisterURL

Registers an application, designated by URL, in the Launch Services database.

```
OSStatus LSRegisterURL (
   CFURLRef inURL,
   Boolean inUpdate
);
```

Parameters

inFileURL

A Core Foundation URL reference designating the application to be registered; see the *CFURL Reference* in the Core Foundation Reference Documentation for a description of the `CFURLRef` data type. The URL must have scheme `file` and contain a valid path to an application file or application bundle.

inUpdate

A Boolean value specifying whether Launch Services should update existing information registered for the application, if any. If this parameter is `false`, the application will not be registered if it has already been registered previously and its current modification date has not changed from when it was last registered; if the parameter is `true`, the application's registered information will be updated even if its modification date has not changed.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
This function adds the designated application and its document and URL claims (if any) to the Launch Services database, making the application a candidate for document and URL binding.

Version Notes
Thread-safe since Mac OS version 10.3.

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

Declared In
`LSInfo.h`

LSSetDefaultHandlerForURLScheme

Sets the user's preferred default handler for the specified URL scheme.

```
OSStatus LSSetDefaultHandlerForURLScheme (
   CFStringRef inURLScheme,
   CFStringRef inHandlerBundleID
);
```

Parameters

inURLScheme

The URL scheme for which the handler is to be set.

inHandlerBundleID

The application bundle identifier that is to be set as the handler for the URL scheme specified by `inURLScheme`.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
Call `LSCopyDefaultHandlerForURLScheme` (page 1207) to get the current setting of the user's preferred default handler for a specified content type.

URL handling capability is determined according to the value of the `CFBundleURLTypes` key in an application's `Info.plist`. For information on the `CFBundleURLTypes` key, see the section "CFBundleURLTypes" in *Mac OS X Runtime Configuration Guidelines*.

Version Notes
Thread-safe since Mac OS X v10.4.

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

Declared In
`LSInfo.h`

LSSetDefaultRoleHandlerForContentType

Sets the user's preferred default handler for the specified content type in the specified roles.

```
OSStatus LSSetDefaultRoleHandlerForContentType (
   CFStringRef inContentType,
   LSRolesMask inRole,
   CFStringRef inHandlerBundleID
);
```

Parameters

inContentType

The content type for which the default role handler is being set. The content type is a uniform type identifier (UTI).

inRole

The roles for which the default role handler is being set. Pass `kLSRolesAll` to specify all roles. For additional possible values, see "Roles Mask" (page 1241).

inHandlerBundleID

The application bundle identifier that is to be set as the default handler for the specified content type and roles.

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Discussion
Call `LSCopyDefaultRoleHandlerForContentType` (page 1208) to get the current setting of the user's preferred default handler for a specified content type.

Version Notes
Thread-safe since Mac OS X v10.4.

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

Declared In
`LSInfo.h`

LSSetExtensionHiddenForRef

Specifies whether the filename extension for an item designated by file-system reference should be hidden or shown.

```
OSStatus LSSetExtensionHiddenForRef (
   const FSRef *inRef,
   Boolean inHide
);
```

Parameters

inRef

A pointer to a file-system reference designating the item whose filename extension is to be hidden or shown; see the *File Manager Reference* in the Carbon File Management Documentation for a description of the `FSRef` data type.

inHide

A Boolean value specifying whether the filename extension should be hidden (`true`) or shown (`false`).

Return Value
A result code; see "Launch Services Result Codes" (page 1251). The function will return the result code `kLSCannotSetInfoErr` if:

- The extension is not valid (contains spaces)
- The extension is not active (is not claimed by an application registered with Launch Services)
- Hiding the extension would make the filename appear to have an active but incorrect extension (for example, in the filename `Photo.jpeg.scpt`, where hiding the extension would make an AppleScript file appear to be a JPEG file)

Discussion
This function sets the necessary file-system state controlling whether the filename extension should be hidden in the display name of the item designated by the `inRef` parameter. To determine whether an item's extension is currently hidden, you can use the `LSCopyItemInfoForRef` function.

Version Notes
Thread-safe since Mac OS version 10.2.

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

Declared In
`LSInfo.h`

LSSetExtensionHiddenForURL

Specifies whether the filename extension for an item designated by URL should be hidden or shown.

```
OSStatus LSSetExtensionHiddenForURL (
   CFURLRef inURL,
   Boolean inHide
);
```

Parameters

`inFileURL`

A Core Foundation URL reference designating the item whose filename extension is to be hidden or shown; see the *CFURL Reference* in the Core Foundation Reference Documentation for a description of the `CFURLRef` data type. The URL must have scheme `file` and contain a valid path to either a file or a directory.

`inHide`

A Boolean value specifying whether the extension should be hidden (`true`) or shown (`false`).

Return Value
A result code; see "Launch Services Result Codes" (page 1251). The function will return the result code `kLSCannotSetInfoErr` if:

- The extension is not valid (contains spaces)
- The extension is not active (is not claimed by an application registered with Launch Services)
- Hiding the extension would make the filename appear to have an active but incorrect extension (for example, in the filename `Photo.jpeg.scpt`, where hiding the extension would make an AppleScript file appear to be a JPEG file)

Discussion
This function sets the necessary file-system state controlling whether the filename extension should be hidden in the display name of the item designated by the `inFileURL` parameter. To determine whether an item's extension is currently hidden, you can use the `LSCopyItemInfoForURL` function.

Version Notes
Thread-safe since Mac OS version 10.2.

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

Declared In
`LSInfo.h`

LSSetHandlerOptionsForContentType

Sets the handler option for the specified content type.

```
OSStatus LSSetHandlerOptionsForContentType (
   CFStringRef inContentType,
   LSHandlerOptions inOptions
);
```

Parameters

inContentType

The content type for which the handler options are to be set. The content type is a uniform type identifier (UTI).

inOptions

The handler options to set. For possible values, see "Handler Option Constants" (page 1249).

Return Value
A result code; see "Launch Services Result Codes" (page 1251).

Version Notes
Thread-safe since Mac OS X v10.4.

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

Declared In
`LSInfo.h`

LSTerm

(Deprecated in Mac OS X v10.3. Formerly used to terminate Launch Services; now does nothing.)

Not recommended.

```
OSStatus LSTerm (
   void
);
```

Discussion
Calling this function was formerly required in order to terminate Launch Services; however, it is no longer needed and so should not be called. It now does nothing at all.

Version Notes
Thread-safe since Mac OS version 10.2.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.

Declared In
`LSInfo.h`

Data Types

This section describes the data types defined in the Launch Services API.

LSApplicationParameters

Specifies the application, launch flags, and additional parameters that control how an application is launched.

```
struct LSApplicationParameters {
    CFIndex version;
    LSLaunchFlags flags;
    const FSRef * application;
    void * asyncLaunchRefCon;
    CFDictionaryRef environment;
    CFArrayRef argv;
    AppleEvent * initialEvent
};
typedef struct LSApplicationParameters LSApplicationParameters;
```

Fields

`version`
The version of the structure. The value of this field must be `0`.

`flags`
Launch flags. For possible values, see "Launch Flags" (page 1242).

`application`
The `FSRef` of the application to open.

`asyncLaunchRefCon`
The client `refCon` that is to appear in subsequent launch notifications.

`environment`
A dictionary of `CFStringRef` keys and values for environment variables to set in the launched process. The value of this field can be `NULL`.

`argv`
An array of values of type `CFStringRef` that specify the arguments that are to be passed to `main()` in the launched process. The value of this field can be `NULL`. This field is ignored in Mac OS X v10.4.

`initialEvent`
The first Apple Event to send to the launched process. The value of this field can be `NULL`.

Discussion

This structure is passed as a parameter to `LSOpenApplication` (page 1223), `LSOpenItemsWithRole` (page 1229), and `LSOpenURLsWithRole` (page 1230).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`LSOpen.h`

LSLaunchFSRefSpec

Specifies, by file-system reference, an application to launch, items to open, or both, along with related information.

```
struct LSLaunchFSRefSpec {
    const FSRef *appRef;
    UInt32 numDocs;
    const FSRef *itemRefs;
    const AEDesc *passThruParams;
    LSLaunchFlags launchFlags;
    void *asyncRefCon;
};
typedef struct LSLaunchFSRefSpec LSLaunchFSRefSpec;
```

Fields

`appRef`

A pointer to a file-system reference designating the application to launch; see the *File Manager Reference* in the Carbon File Management Documentation for a description of the `FSRef` data type. Set this field to `NULL` to request that each item in the `itemRefs` array be opened in its own preferred application.

`numDocs`

The number of elements in the array specified by the `itemRefs` field. The value of this field can be `0`, in which case the application designated by `appRef` is launched without opening any items.

`itemRefs`

An array of file-system references designating the item or items to open. If the value of `numDocs` is `0`, this field is ignored and can be set to `NULL`.

`passThruParams`

A pointer to an Apple event descriptor that is passed untouched as an optional parameter, with keyword `keyAEPropData` (`'prdt'`), in the Apple event sent to each application launched or activated (whether individual preferred applications or the application designated by `appRef`). See the *Apple Event Manager Reference* in the Carbon Interapplication Communication Documentation for a description of the `AEDesc` data type. The value of this field can be `NULL`.

`launchFlags`

Launch flags specifying how to launch each application (including whether to print or merely open documents); see "Launch Flags" (page 1242) for a description of these flags.

`asyncRefCon`

A pointer to an arbitrary application-defined value, passed in the Carbon event notifying you of an application's launch or termination (if you have registered for such notification). The value of this field can be `NULL`.

Discussion

This data type defines a file-based launch specification designating, by file-system reference, an application to launch, items to open, or both. To request that items be opened in a particular application, set `appRef`, `numDocs`, and `itemRefs` accordingly. To request that each designated item be opened in its own preferred application, set *`appRef`* to `NULL`. To request that a particular application be launched without opening any documents, set *`appRef`* accordingly and set `numDocs` to `0`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`LSOpen.h`

LSLaunchURLSpec

Specifies, by URL, an application to launch, items to open, or both, along with related information.

```
struct LSLaunchURLSpec {
    CFURLRef appURL;
    CFArrayRef itemURLs;
    const AEDesc *passThruParams;
    LSLaunchFlags launchFlags;
    void *asyncRefCon;
};
typedef struct LSLaunchURLSpec LSLaunchURLSpec;
```

Fields

`appURL`

A Core Foundation URL reference designating the application to launch; see the *CFURL Reference* in the Core Foundation Reference Documentation for a description of the `CFURLRef` data type. The URL must have scheme `file` and contain a valid path to an application file or application bundle. Set this field to `NULL` to request that each item in the `itemURLs` array be opened in its own preferred application.

`itemURLs`

A reference to an array of Core Foundation URL references designating the item or items to open; see the *CFArray Reference* in the Core Foundation Reference Documentation for a description of the `CFArrayRef` data type. The value of this field can be `NULL`, in which case the application designated by `appURL` will be launched without opening any items.

`passThruParams`

A pointer to an Apple event descriptor that is passed untouched as an optional parameter, with keyword `keyAEPropData` (`'prdt'`), in the Apple event sent to each application launched or activated (whether individual preferred applications or the application designated by `appURL`). See the *Apple Event Manager Reference* in the Carbon Interapplication Communication Documentation for a description of the `AEDesc` data type. The value of this field can be `NULL`.

`launchFlags`

Launch flags specifying how to launch each application (including whether to print or merely open documents); see "Launch Flags" (page 1242) for a description of these flags.

`asyncRefCon`

A pointer to an arbitrary application-defined value, passed in the Carbon event notifying you of an application's launch or termination (if you have registered for such notification). The value of this field can be `NULL`.

Discussion

This data type defines a URL-based launch specification designating, by URL, an application to launch, items to open, or both. To request that items be opened in a particular application, set *`appURL`* and `itemURLs` accordingly. To request that each designated item be opened in its own preferred application, set *`appURL`* to `NULL`. If the item URL's scheme is `file` (designating either a file or a directory), the selection of the preferred application is based on the designated item's filename extension, file type, and creator signature; otherwise, it is based on the URL scheme (such as `http`, `ftp`, or `mailto`). To request that a particular application be launched without opening any document, set *`appURL`* accordingly and set `itemURLs` to `NULL`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`LSOpen.h`

LSItemInfoRecord

Contains requested information about an item.

```
struct LSItemInfoRecord {
    LSItemInfoFlags flags;
    OSType filetype;
    OSType creator;
    CFStringRef extension;
    CFStringRef iconFileName;
    LSKindID kindID;
};
typedef struct LSItemInfoRecord LSItemInfoRecord;
```

Fields

`flags`
: Item-information flags specifying information about the item; see "Item-Information Flags" (page 1246) for a description of these flags.

`filetype`
: The item's file type.

`creator`
: The item's creator signature.

`extension`
: A Core Foundation string object specifying the item's filename extension; see the *CFString Reference* in the Core Foundation Reference Documentation for a description of the `CFStringRef` data type.

`iconFileName`
: No longer used.

`kindID`
: No longer used.

Discussion

This data type defines an item-information record used by the `LSCopyItemInfoForRef` and `LSCopyItemInfoForURL` functions to return requested information about an item.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`LSInfo.h`

LSKindID

Data type of the `kindID` field of an item-information record (`LSItemInfoRecord`); no longer used.

```
typedef UInt32 LSKindID;
```

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`LSInfo.h`

Constants

This section describes the constants defined in the Launch Services API.

Roles Mask

Specify the desired role or roles for an application to claim with respect to an item or family of items.

```
typedef OptionBits LSRolesMask;enum {
    kLSRolesNone = 0x00000001,
    kLSRolesViewer = 0x00000002,
    kLSRolesEditor = 0x00000004,
    kLSRolesShell = 0x00000008,
    kLSRolesAll = 0xFFFFFFFF
};
```

Constants

`kLSRolesNone`

Requests the role `None` (the application cannot open the item, but provides an icon and a kind string for it).

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSRolesViewer`

Requests the role `Viewer` (the application can read and present the item, but cannot manipulate or save it).

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSRolesEditor`

Requests the role `Editor` (the application can read, present, manipulate, and save the item).

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSRolesShell`

Requests the role `Shell` (the application can execute the item).

Available in Mac OS X v10.4 and later.

Declared in `LSInfo.h`.

`kLSRolesAll`

Accepts any role with respect to the item.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

Discussion

This bit mask is passed to functions that find the preferred application for a given item or family of items (`LSGetApplicationForItem`, `LSGetApplicationForURL`, `LSGetApplicationForInfo`), or that determine whether a given application can open a designated item (`LSCanRefAcceptItem`, `LSCanURLAcceptURL`), to specify the application's desired role or roles with respect to the item. For example, to request only an editor application, specify `kLSRolesEditor`; if either an editor or a viewer application is acceptable, specify `kLSRolesEditor | kLSRolesViewer`.

Launch Flags

Specify how to launch an application.

```
typedef OptionBits LSLaunchFlags;enum {
    kLSLaunchDefaults = 0x00000001,
    kLSLaunchAndPrint = 0x00000002,
    kLSLaunchReserved2 = 0x00000004,
    kLSLaunchReserved3 = 0x00000008,
    kLSLaunchReserved4 = 0x00000010,
    kLSLaunchReserved5 = 0x00000020,
    kLSLaunchAndDisplayErrors = 0x00000040,
    kLSLaunchInhibitBGOnly = 0x00000080,
    kLSLaunchDontAddToRecents = 0x00000100,
    kLSLaunchDontSwitch = 0x00000200,
    kLSLaunchNoParams = 0x00000800,
    kLSLaunchAsync = 0x00010000,
    kLSLaunchStartClassic = 0x00020000,
    kLSLaunchInClassic = 0x00040000,
    kLSLaunchNewInstance = 0x00080000,
    kLSLaunchAndHide = 0x00100000,
    kLSLaunchAndHideOthers = 0x00200000,
    kLSLaunchHasUntrustedContents = 0x00400000
};
```

Constants

`kLSLaunchDefaults`

Requests launching in the default manner (as if the only flags set were `kLSLaunchNoParams`, `kLSLaunchAsync`, and `kLSLaunchStartClassic`).

Available in Mac OS X v10.0 and later.

Declared in `LSOpen.h`.

`kLSLaunchAndPrint`

Requests that documents opened in the application be printed.

Available in Mac OS X v10.0 and later.

Declared in `LSOpen.h`.

`kLSLaunchReserved2`

Reserved for future use.

Available in Mac OS X v10.0 and later.

Declared in `LSOpen.h`.

`kLSLaunchReserved3`

Reserved for future use.

Available in Mac OS X v10.0 and later.

Declared in `LSOpen.h`.

`kLSLaunchReserved4`

Reserved for future use.

Available in Mac OS X v10.0 and later.

Declared in `LSOpen.h`.

`kLSLaunchReserved5`
Reserved for future use.
Available in Mac OS X v10.0 and later.
Declared in `LSOpen.h`.

`kLSLaunchAndDisplayErrors`
Requests that launch and open failures be displayed in the UI.
Available in Mac OS X v10.4 and later.
Declared in `LSOpen.h`.

`kLSLaunchInhibitBGOnly`
Requests that the launch be made to fail if the application is background-only.
Available in Mac OS X v10.0 and later.
Declared in `LSOpen.h`.

`kLSLaunchDontAddToRecents`
Requests that the application or documents not be added to the Finder's Recent Items menu.
Available in Mac OS X v10.0 and later.
Declared in `LSOpen.h`.

`kLSLaunchDontSwitch`
Requests that the application be launched without being brought to the foreground.
Available in Mac OS X v10.0 and later.
Declared in `LSOpen.h`.

`kLSLaunchNoParams`
Requests that the application's information property list be used to determine the launch parameters.
Available in Mac OS X v10.0 and later.
Declared in `LSOpen.h`.

`kLSLaunchAsync`
Requests that the application be launched asynchronously: that is, the Launch Services function launching it return control immediately, without waiting for it to complete its launch sequence (indicated visually to the user when the application's icon stops "bouncing" in the Dock).
Available in Mac OS X v10.0 and later.
Declared in `LSOpen.h`.

`kLSLaunchStartClassic`
Requests that the Classic emulation environment be started up if the application requires it. If this flag is not set and the application requires the Classic environment, the launch will fail.
Available in Mac OS X v10.0 and later.
Declared in `LSOpen.h`.

`kLSLaunchInClassic`
Requests that the application be forced to launch in the Classic emulation environment.
Available in Mac OS X v10.0 and later.
Declared in `LSOpen.h`.

`kLSLaunchNewInstance`
Requests that a new instance of the application be started, even if one is already running.
Available in Mac OS X v10.0 and later.
Declared in `LSOpen.h`.

`kLSLaunchAndHide`
Requests that the application be hidden as soon as it completes its launch sequence.

Available in Mac OS X v10.0 and later.

Declared in `LSOpen.h`.

`kLSLaunchAndHideOthers`
Requests that other applications be hidden as soon as the opened application completes its launch sequence.

Available in Mac OS X v10.0 and later.

Declared in `LSOpen.h`.

`kLSLaunchHasUntrustedContents`
Requests that the items to be launched should be marked as untrusted.

Available in Mac OS X v10.4 and later.

Declared in `LSOpen.h`.

Discussion
They are passed in a launch specification structure (`LSLaunchFSRefSpec` to the `LSOpenFromRefSpec` function or `LSLaunchURLSpec` to the `LSOpenFromURLSpec` function), to control the manner in which applications are launched.

Requested-Information Flags

Specify what information to obtain about an item.

```
typedef OptionBits LSRequestedInfo;enum {
    kLSRequestExtension = 0x00000001,
    kLSRequestTypeCreator = 0x00000002,
    kLSRequestBasicFlagsOnly = 0x00000004,
    kLSRequestAppTypeFlags = 0x00000008,
    kLSRequestAllFlags = 0x00000010,
    kLSRequestIconAndKind = 0x00000020,
    kLSRequestExtensionFlagsOnly = 0x00000040,
    kLSRequestAllInfo = 0xFFFFFFFF
};
```

Constants
`kLSRequestExtension`
Requests the item's filename extension.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSRequestTypeCreator`
Requests the item's file type and creator signature.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSRequestBasicFlagsOnly`
Requests all item-information flags that are not application-specific: that is, all except `kLSItemInfoIsNativeApp` through `kLSItemInfoAppIsScriptable`.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSRequestAppTypeFlags`
Requests all application-specific item-information flags: that is, `kLSItemInfoIsNativeApp` through `kLSItemInfoAppIsScriptable`.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSRequestAllFlags`
Requests all item-information flags.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSRequestIconAndKind`
Not used.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSRequestExtensionFlagsOnly`
Requests only the `kLSItemInfoExtensionIsHidden` item-information flag.

Available in Mac OS X v10.1 and later.

Declared in `LSInfo.h`.

`kLSRequestAllInfo`
Requests all available item information.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

Discussion
These flags are passed to the `LSCopyItemInfoForRef` and `LSCopyItemInfoForURL` functions to specify the type of information to be obtained in an item-information record; see LSItemInfoRecord (page 1240) for a description of this structure.

Item Attribute Constants

Constants used to retrieve item attributes.

```
const CFStringRef kLSItemContentType;
const CFStringRef kLSItemFileType;
const CFStringRef kLSItemFileCreator;
const CFStringRef kLSItemExtension;
const CFStringRef kLSItemDisplayName
const CFStringRef kLSItemDisplayKind;
const CFStringRef kLSItemRoleHandlerDisplayName;
const CFStringRef kLSItemIsInvisible;
const CFStringRef kLSItemExtensionIsHidden;
```

Constants
`kLSItemContentType`
The item's content type identifier, which is a uniform type identifier string. The value type of this attribute is `CFStringRef`.

Available in Mac OS X v10.4 and later.

Declared in `LSInfo.h`.

`kLSItemFileType`
The item's file type (OSType). The value type of this attribute is `CFStringRef`.

Available in Mac OS X v10.4 and later.

Declared in `LSInfo.h`.

`kLSItemFileCreator`
The item's file creator (OSType). The value type of this attribute is `CFStringRef`.

Available in Mac OS X v10.4 and later.

Declared in `LSInfo.h`.

`kLSItemExtension`
The item's filename extension. The value type of this attribute is `CFStringRef`.

Available in Mac OS X v10.4 and later.

Declared in `LSInfo.h`.

`kLSItemDisplayName`
The item's name as displayed to the user. The display name reflects localization and extension hiding that may be in effect. The value type of this attribute is `CFStringRef`.

Available in Mac OS X v10.4 and later.

Declared in `LSInfo.h`.

`kLSItemDisplayKind`
The localized kind string describing the item's type. The value type of this attribute is `CFStringRef`.

Available in Mac OS X v10.4 and later.

Declared in `LSInfo.h`.

`kLSItemRoleHandlerDisplayName`
The display name of the application that is set to handle this item, subject to the role mask. The value type of this attribute is `CFStringRef`.

Available in Mac OS X v10.4 and later.

Declared in `LSInfo.h`.

`kLSItemIsInvisible`
A value of `kCFBooleanTrue` if the item is normally hidden from users; otherwise, `kCFBooleanFalse`. The value type of this attribute is `CFBooleanRef`.

Available in Mac OS X v10.4 and later.

Declared in `LSInfo.h`.

`kLSItemExtensionIsHidden`
A value of `kCFBooleanTrue` if the item's extension is set to be hidden; otherwise, `kCFBooleanFalse`. The value type of this attribute is `CFBooleanRef`.

Available in Mac OS X v10.4 and later.

Declared in `LSInfo.h`.

Item-Information Flags

Provide information about an item.

```
typedef OptionBits LSItemInfoFlags;enum {
    kLSItemInfoIsPlainFile = 0x00000001,
    kLSItemInfoIsPackage = 0x00000002,
    kLSItemInfoIsApplication = 0x00000004,
    kLSItemInfoIsContainer = 0x00000008,
    kLSItemInfoIsAliasFile = 0x00000010,
    kLSItemInfoIsSymlink = 0x00000020,
    kLSItemInfoIsInvisible = 0x00000040,
    kLSItemInfoIsNativeApp = 0x00000080,
    kLSItemInfoIsClassicApp = 0x00000100,
    kLSItemInfoAppPrefersNative = 0x00000200,
    kLSItemInfoAppPrefersClassic = 0x00000400,
    kLSItemInfoAppIsScriptable = 0x00000800,
    kLSItemInfoIsVolume = 0x00001000,
    kLSItemInfoExtensionIsHidden = 0x00100000
};
```

Constants

`kLSItemInfoIsPlainFile`
Item is a data file (and not, for example, a directory, volume, or UNIX symbolic link).
Available in Mac OS X v10.0 and later.
Declared in `LSInfo.h`.

`kLSItemInfoIsPackage`
Item is a packaged directory.
Available in Mac OS X v10.0 and later.
Declared in `LSInfo.h`.

`kLSItemInfoIsApplication`
Item is a single-file or packaged application.
Available in Mac OS X v10.0 and later.
Declared in `LSInfo.h`.

`kLSItemInfoIsContainer`
Item is a directory (includes packages) or volume.
Available in Mac OS X v10.0 and later.
Declared in `LSInfo.h`.

`kLSItemInfoIsAliasFile`
Item is an alias file (includes symbolic links).
Available in Mac OS X v10.0 and later.
Declared in `LSInfo.h`.

`kLSItemInfoIsSymlink`
Item is a UNIX symbolic link.
Available in Mac OS X v10.0 and later.
Declared in `LSInfo.h`.

`kLSItemInfoIsInvisible`
Item is invisible, because either its name begins with a period or its `isInvisible` Finder flag is set.
Available in Mac OS X v10.0 and later.
Declared in `LSInfo.h`.

`kLSItemInfoIsNativeApp`
Item is an application that can run natively in Mac OS X.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSItemInfoIsClassicApp`
Item is an application that cannot run natively and must be launched in the Classic emulation environment.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSItemInfoAppPrefersNative`
Item is an application that can run either natively or in the Classic emulation environment, but prefers to be launched natively. This flag is valid only when `kLSItemInfoIsNativeApp` is set.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSItemInfoAppPrefersClassic`
Item is an application that can run either natively or in the Classic emulation environment, but prefers to be launched in the Classic environment. This flag is valid only when `kLSItemInfoIsNativeApp` is set.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSItemInfoAppIsScriptable`
Item is an application that can be scripted.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSItemInfoIsVolume`
Item is the root directory of a volume or mount point.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSItemInfoExtensionIsHidden`
Item has a hidden filename extension.

Available in Mac OS X v10.1 and later.

Declared in `LSInfo.h`.

Discussion
These flags are set in an item-information record to provide information about an item; see LSItemInfoRecord (page 1240) for a description of this structure.

Acceptance Flags

Specify behavior to observe when testing whether an application can accept (open) an item.

```
typedef OptionBits LSAcceptanceFlags;enum {
    kLSAcceptDefault = 0x00000001,
    kLSAcceptAllowLoginUI = 0x00000002
};
```

Constants

`kLSAcceptDefault`

Requests the default behavior (as opposed to the behavior specified by the `kLSAcceptAllowLoginUI` flag).

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSAcceptAllowLoginUI`

Requests that the user interface to log in be presented if necessary. If `LSCanRefAcceptItem` or `LSCanURLAcceptURL` is called during a drag-and-drop operation, showing a server login dialog would be an inappropriate user experience. If the target designated in the function call is an alias to an application, Launch Services needs to resolve the alias to ascertain what file types the application can open; however, if the application is on a server that needs to be authenticated, Launch Services will by default fail to resolve the alias, to avoid having to present the login interface. To override this default behavior by allowing the server login interface, set the `kLSAcceptAllowLoginUI` flag.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

Discussion

These flags are passed to the functions `LSCanRefAcceptItem` and `LSCanURLAcceptURL`.

Handler Option Constants

Specify the options for controlling how content handlers are selected.

```
typedef OptionBits LSHandlerOptions;enum {
    kLSHandlerOptionsDefault = 0,
    kLSHandlerOptionsIgnoreCreator = 1
};
```

Constants

`kLSHandlerOptionsDefault`

When set, causes Launch Services to use a content item's creator (when available) to select a handler. This is the default setting.

Available in Mac OS X v10.4 and later.

Declared in `LSInfo.h`.

`kLSHandlerOptionsIgnoreCreator`

When set, causes Launch Services to ignore the content item's creator when selecting a role handler for the specified content type.

Available in Mac OS X v10.4 and later.

Declared in `LSInfo.h`.

Invalid Extension Index

Represents an invalid filename extension index.

```
enum {
    kLSInvalidExtensionIndex = 0xFFFFFFFF
};
```

Constants

`kLSInvalidExtensionIndex`

The value obtained by the `LSGetExtensionInfo` function if the filename does not contain a valid extension.

Available in Mac OS X v10.1 through Mac OS X v10.4.

Declared in `LSInfo.h`.

Unknown Type or Creator

Represent an unknown file type or creator.

```
enum {
    kLSUnknownType = 0,
    kLSUnknownCreator = 0
};
```

Constants

`kLSUnknownType`

The value to supply as the file type (for example, to the `LSGetApplicationForInfo` function) if no file type information is available.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSUnknownCreator`

The value to supply as the creator signature if no file creator information is available.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

Constants No Longer Used

The following constants are no longer used.

```
typedef OptionBits LSInitializeFlags;enum {
    kLSInitializeDefaults = 0x00000001
};enum {
    kLSUnknownKindID = 0
};enum {
    kLSMinCatInfoBitmap = (kFSCatInfoNodeFlags | kFSCatInfoParentDirID
| kFSCatInfoFinderInfo | kFSCatInfoFinderXInfo)
};
```

Constants

`kLSInitializeDefaults`

Formerly passed to the `LSInit` function, which is no longer used.

Available in Mac OS X v10.0 and later.

Declared in `LSInfo.h`.

`kLSUnknownKindID`

A possible value of the `kindID` field of an item-information record, which is no longer used.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `LSInfo.h`.

`kLSMinCatInfoBitmap`

A minimal catalog information bitmap; no longer used.

Available in Mac OS X v10.1 and later.

Declared in `LSInfo.h`.

Result Codes

The table below lists the most common result codes returned by Launch Services functions.

Result Code	Value	Description
`kLSAppInTrashErr`	-10660	The application cannot be run because it is inside a Trash folder. Available in Mac OS X v10.3 and later.
`kLSUnknownErr`	-10810	An unknown error has occurred. Available in Mac OS X v10.0 and later.
`kLSNotAnApplicationErr`	-10811	The item to be registered is not an application. Available in Mac OS X v10.0 and later.
`kLSNotInitializedErr`	-10812	Formerly returned by `LSInit` on initialization failure; no longer used. Available in Mac OS X v10.0 and later.
`kLSDataUnavailableErr`	-10813	Data of the desired type is not available (for example, there is no kind string). Available in Mac OS X v10.0 and later.
`kLSApplicationNotFoundErr`	-10814	No application in the Launch Services database matches the input criteria. Available in Mac OS X v10.0 and later.
`kLSUnknownTypeErr`	-10815	Not currently used. Available in Mac OS X v10.0 and later.
`kLSDataTooOldErr`	-10816	Not currently used. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kLSDataErr	-10817	Data is structured improperly (for example, an item's information property list is malformed). Not used in Mac OS X v10.4. Available in Mac OS X v10.0 and later.
kLSLaunchInProgressErr	-10818	A launch of the application is already in progress. Available in Mac OS X v10.0 and later.
kLSNotRegisteredErr	-10819	Not currently used. Available in Mac OS X v10.0 and later.
kLSAppDoesNotClaimTypeErr	-10820	Not currently used. Available in Mac OS X v10.0 and later.
kLSAppDoesNotSupportSchemeWarning	-10821	Not currently used. Available in Mac OS X v10.0 and later.
kLSServerCommunicationErr	-10822	There is a problem communicating with the server process that maintains the Launch Services database. Available in Mac OS X v10.0 and later.
kLSCannotSetInfoErr	-10823	The filename extension to be hidden cannot be hidden. Available in Mac OS X v10.1 and later.
kLSNoRegistrationInfoErr	-10824	Not currently used. Available in Mac OS X v10.3 and later.
kLSIncompatibleSystemVersionErr	-10825	The application to be launched cannot run on the current Mac OS version. Available in Mac OS X v10.3 and later.
kLSNoLaunchPermissionErr	-10826	The user does not have permission to launch the application (on a managed network). Available in Mac OS X v10.3 and later.
kLSNoExecutableErr	-10827	The executable file is missing or has an unusable format. Available in Mac OS X v10.3 and later.
kLSNoClassicEnvironmentErr	-10828	The Classic emulation environment was required but is not available. Available in Mac OS X v10.3 and later.

Result Code	Value	Description
kLSMultipleSessionsNotSupportedErr	-10829	The application to be launched cannot run simultaneously in two different user sessions. Available in Mac OS X v10.3 and later.

Locale Utilities Reference

Framework: CoreServices/CoreServices.h

Declared in MacLocales.h

Overview

Unicode operations such as collation and text boundary determination can be affected by the conventions of a particular language or region. You can use Locale Utilities to specify language-specific or region-specific information for locale-sensitive Unicode operations. The Locale Utilities provide support for obtaining available locales, obtaining localized locale names, and converting among locale data formats.

Carbon supports the Locales Utilities.

Functions by Task

Obtaining Available Locales

`LocaleOperationGetLocales` (page 1263)
Obtains the list of available locale-and-variant combinations for a given class of operation.

`LocaleOperationCountLocales` (page 1261)
Obtains the total count of locale-and-variant combinations available for a given class of operation.

Obtaining Localized Locale Names

`LocaleGetName` (page 1258)
Obtains the localized name for a locale and/or operation variant, based on the requested language for the localized name.

`LocaleGetIndName` (page 1257)
Obtains a localized name for a locale and/or operation variant, based on an index into the list of all available localized names.

`LocaleCountNames` (page 1256)
Obtains the total count of all available localized names for a locale and/or operation variant.

`LocaleOperationGetName` (page 1264)
Obtains the localized name for an operation class, based on the requested language for the localized name.

`LocaleOperationGetIndName` (page 1262)
Obtains a localized name for an operation class, based on an index into the list of all available localized names.

`LocaleOperationCountNames` (page 1261)
Obtains the total count of all available localized names for an operation class.

`LocaleGetRegionLanguageName` (page 1260) Deprecated in Mac OS X v10.5
Obtains the localized language name for a region.

Converting Among Locale Data Formats

`LocaleRefFromLangOrRegionCode` (page 1265)
Converts a Mac OS language or region code to a locale reference.

`LocaleRefFromLocaleString` (page 1266)
Converts a string containing locale data to a locale reference.

`LocaleRefGetPartString` (page 1267)
Converts a locale reference to a string containing locale data.

`LocaleStringToLangAndRegionCodes` (page 1268)
Converts a string containing locale data to Mac OS language and region codes.

Functions

LocaleCountNames

Obtains the total count of all available localized names for a locale and/or operation variant.

```
OSStatus LocaleCountNames (
   LocaleRef locale,
   LocaleOperationVariant opVariant,
   LocaleNameMask nameMask,
   ItemCount *nameCount
);
```

Parameters

locale
A `LocaleRef` value identifying the locale for which you are requesting a localized name.

opVariant
A `LocaleOperationVariant` value identifying the operation variant for which you are requesting a localized name. Pass 0 to obtain the default operation variant.

nameMask
A `LocaleNameMask` value specifying the parts of the name that you are requesting: the name of the locale alone, the name of the operation variant alone, or the name for the combination of the two.

nameCount
A pointer to an `ItemCount` value. On return, the value is set to the total count of all available localized names for the locale and/or operation variant.

Return Value
A result code. The `LocaleCountNames` function can return `paramErr` if `nameCount` is `NULL`.

Discussion
This function provides a count of all available localized names for a given locale, operation variant, or locale-and-variant combination, for use in the function `LocaleGetIndName` (page 1257). You must call this function prior to calling `LocaleGetIndName`.

Special Considerations

This function can move memory in Carbon and Mac OS 9.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.
Available in Mac OS X 10.0 and later.

Declared In
`MacLocales.h`

LocaleGetIndName

Obtains a localized name for a locale and/or operation variant, based on an index into the list of all available localized names.

```
OSStatus LocaleGetIndName (
   LocaleRef locale,
   LocaleOperationVariant opVariant,
   LocaleNameMask nameMask,
   ItemCount nameIndex,
   UniCharCount maxNameLen,
   UniCharCount *actualNameLen,
   UniChar displayName[],
   LocaleRef *displayLocale
);
```

Parameters

`locale`
A `LocaleRef` value identifying the locale for which you are requesting a localized name.

`opVariant`
A `LocaleOperationVariant` value identifying the operation variant for which you are requesting a localized name. Pass 0 to obtain the default operation variant.

`nameMask`
A `LocaleNameMask` value specifying the parts of the name that you are requesting: the name of the locale alone, the name of the operation variant alone, or the name for the combination of the two.

`nameIndex`
An `ItemCount` index value identifying an entry in the list of available localized names. You can obtain a count of all available localized names from the `nameCount` parameter of the function `LocaleCountNames` (page 1256). Note that index values must be in the range from 0 to `nameCount` -1. To create a list of all available localized names that correspond to the locale and operation variant you request, you can call the `LocaleGetIndName` function starting with an index value of 0, then increment the index value with each successive call to `LocaleGetIndName`, continuing to `nameCount`-1.

maxNameLen

A UniCharCount value specifying the length of the UniChar array being passed in the displayName parameter. An array of 32 characters is typically adequate if you request just a single name (that is, the name for a locale or an operation variant), or 64 characters if you request both names.

actualNameLen

A pointer to a UniCharCount value. On return, the value contains the actual length of the name produced in the displayName parameter.

displayName

An array of UniChar values. On return, the array contains the requested name as localized for a particular display locale.

displayLocale

A pointer to a LocaleRef value. On return, the value indicates the language of the name produced in the displayName parameter.

Return Value

A result code. If the value specified in the maxNameLen parameter is too small for the requested string, LocaleGetIndName returns kLocalesBufferTooSmallErr (-30001).

Discussion

The LocaleGetIndName function obtains a localized name for a locale, operation variant, or locale-and-variant combination. LocaleGetIndName produces the name based on a requested locale, an operation variant, and an index into the count of all available localized names for the locale and/or operation variant. The language for the obtained name is produced on return.

You can use repeated calls to the LocaleGetIndName function to create a list of all available localized names (that is, the names from all available display locales), that correspond to a given locale and operation variant. Alternately, for a particular locale, operation variant, or locale-and-variant combination, the function LocaleGetName (page 1258) obtains the corresponding name as localized for a particular display locale.

Note that, in some languages, the name for a language or locale may have several different grammatical forms, where the correct form depends on the usage or context. For example, Swedish uses different forms of a language name depending on whether the name is applied to a collation order, to text break rules, or to keyboard layouts. However, the Locale Utilities functions only return one form of a language or locale name—typically that name which would be used for the language name alone. Therefore, this name may not be the correct form for some usages in some languages.

Special Considerations

This function can move memory in Carbon and Mac OS 9.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.

Available in Mac OS X 10.0 and later.

Declared In

MacLocales.h

LocaleGetName

Obtains the localized name for a locale and/or operation variant, based on the requested language for the localized name.

```
OSStatus LocaleGetName (
   LocaleRef locale,
   LocaleOperationVariant opVariant,
   LocaleNameMask nameMask,
   LocaleRef displayLocale,
   UniCharCount maxNameLen,
   UniCharCount *actualNameLen,
   UniChar displayName[]
);
```

Parameters

locale

A `LocaleRef` value identifying the locale for which you are requesting a localized name.

opVariant

A `LocaleOperationVariant` value identifying the operation variant for which you are requesting a localized name. Pass 0 to obtain the default operation variant.

nameMask

A `LocaleNameMask` value specifying the parts of the name that you are requesting: the name of the locale alone, the name of the operation variant alone, or the name for the combination of the two.

displayLocale

A `LocaleRef` value indicating the requested language for the name produced in the `displayName` parameter. If `LocaleGetName` cannot find a name in the requested language, it produces a name in the default display locale.

maxNameLen

A `UniCharCount` value specifying the length of the `UniChar` array being passed in the `displayName` parameter. An array of 32 characters is typically adequate if you request just a single name (that is, the name for a locale or an operation variant), or 64 characters if you request both names.

actualNameLen

A pointer to a `UniCharCount` value. On return, the value contains the actual length of the name produced in the `displayName` parameter.

displayName

An array of `UniChar` values. On return, the array contains the requested name as localized for a particular display locale.

Return Value

A result code. If `LocaleGetName` cannot find a name that matches the language specified in the `displayLocale` parameter, it returns `kLocalesDefaultDisplayStatus` (-30029). If the value specified in the `maxNameLen` parameter is too small for the requested string, `LocaleGetName` returns `kLocalesBufferTooSmallErr` (-30001).

Discussion

For a particular locale, operation variant, or locale-and-variant combination, the `LocaleGetName` function obtains the corresponding name as localized for a particular display locale. Alternately, you can use repeated calls to the function `LocaleGetIndName` (page 1257) to iterate through all of the available display locales to create a list of the corresponding names as localized in each of the display locales.

Note that, in some languages, the name for a language or locale may have several different grammatical forms, where the correct form depends on the usage or context. For example, Swedish uses different forms of a language name depending on whether the name is applied to a collation order, to text break rules, or to keyboard layouts. However, the Locale Utilities functions only return one form of a language or locale name—typically that name which would be used for the language name alone. Therefore, this name may not be the correct form for some usages in some languages.

Special Considerations

This function can move memory in Carbon and Mac OS 9.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.
Available in Mac OS X 10.0 and later.

Declared In

`MacLocales.h`

LocaleGetRegionLanguageName

Obtains the localized language name for a region. (Deprecated in Mac OS X v10.5.)

```
OSStatus LocaleGetRegionLanguageName (
   RegionCode region,
   Str255 languageName
);
```

Parameters

region

A Mac OS `RegionCode` value, specifying the region for which you are requesting a localized language name.

languageName

A Pascal string. On return, the string contains the name of the language corresponding to the region specified in the `region` parameter. The language name is produced in its own language and in the appropriate Mac OS encoding for that region.

Return Value

A result code. The `LocaleGetRegionLanguageName` function returns `paramErr` if `languageName` is `NULL` or if `region` is invalid (

Discussion

For a particular Mac OS region code, the `LocaleGetRegionLanguageName` function returns the name of the corresponding language in that language and in the non-Unicode Mac OS text encoding used for that region.

Note that, in some languages, the name for a language or locale may have several different grammatical forms, where the correct form depends on the usage or context. For example, Swedish uses different forms of a language name depending on whether the name is applied to a collation order, to text break rules, or to keyboard layouts. However, the Locale Utilities functions only return one form of a language or locale name—typically that name which would be used for the language name alone. Therefore, this name may not be the correct form for some usages in some languages.

Special Considerations

This function can move memory in Carbon and Mac OS 9.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 9 or 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
MacLocales.h

LocaleOperationCountLocales

Obtains the total count of locale-and-variant combinations available for a given class of operation.

```
OSStatus LocaleOperationCountLocales (
   LocaleOperationClass opClass,
   ItemCount *localeCount
);
```

Parameters

opClass

A LocaleOperationClass value identifying the operation class for which you are requesting availability information.

localeCount

A pointer to an ItemCount value. On return, the value is set to the total count of all available locale-and-variant combinations for the specified class of operation.

Return Value
A result code. The LocaleOperationCountLocales function returns paramErr if the opClass parameter is 0 or if the localeCount parameter is NULL.

Discussion
The LocaleOperationCountLocales function counts the total number of locale-and-variant combinations available for a given operation class. You should call LocaleOperationCountLocales before the function LocaleOperationGetLocales (page 1263) in order to determine how much memory to allocate for the list LocaleOperationGetLocales produces.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.
Available in Mac OS X 10.0 and later.

Declared In
MacLocales.h

LocaleOperationCountNames

Obtains the total count of all available localized names for an operation class.

```
OSStatus LocaleOperationCountNames (
   LocaleOperationClass opClass,
   ItemCount *nameCount
);
```

Parameters

opClass

A LocaleOperationClass value identifying the operation class for which you are requesting a localized name.

nameCount

A pointer to an `ItemCount` value. On return, the value is set to the total count of all available localized names for the operation class.

Return Value

A result code. The `LocaleOperationCountNames` function returns `paramErr` if `nameCount` is `NULL`.

Discussion

This function provides a count of all available localized names for a given operation class, for use in the function `LocaleOperationGetIndName` (page 1262). You must call this function prior to calling `LocaleOperationGetIndName`.

Special Considerations

This function can move memory in Carbon and Mac OS 9.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.

Available in Mac OS X 10.0 and later.

Declared In

`MacLocales.h`

LocaleOperationGetIndName

Obtains a localized name for an operation class, based on an index into the list of all available localized names.

```
OSStatus LocaleOperationGetIndName (
   LocaleOperationClass opClass,
   ItemCount nameIndex,
   UniCharCount maxNameLen,
   UniCharCount *actualNameLen,
   UniChar displayName[],
   LocaleRef *displayLocale
);
```

Parameters

opClass

A `LocaleOperationClass` value identifying the operation class for which you are requesting a localized name.

nameIndex

An `ItemCount` index value identifying an entry in the list of available localized names. You can obtain a count of all available localized names from the `nameCount` parameter of the function `LocaleOperationCountNames` (page 1261). Note that index values must be in the range from 0 to `nameCount` -1. To create a list of all available localized names that correspond to the requested operation class, you can call the `LocaleOperationGetIndName` function starting with an index value of 0, then increment the index value with each successive call to `LocaleOperationGetIndName`, continuing to `nameCount`-1.

maxNameLen

A `UniCharCount` value specifying the length of the `UniChar` array being passed in the `displayName` parameter.

actualNameLen
A pointer to a `UniCharCount` value. On return, the value contains the actual length of the name produced in the `displayName` parameter.

displayName
An array of `UniChar` values. On return, the array contains the requested name as localized for a particular display locale.

displayLocale
A pointer to a `LocaleRef` value. On return, the value indicates the language of the name produced in the `displayName` parameter.

Return Value
A result code. If the value specified in the `maxNameLen` parameter is too small for the requested string, `LocaleOperationGetIndName` returns `kLocalesBufferTooSmallErr` (-30001).

Discussion
The `LocaleOperationGetIndName` function obtains a localized name for an operation class based on an index into the count of all available localized names for the operation class. The language for the obtained name is produced on return.

You can use repeated calls to the `LocaleOperationGetIndName` function to create a list of all available localized names (that is, the names from all available display locales), that correspond to a given operation class. Alternately, for a particular operation class, the function `LocaleOperationGetName` (page 1264) obtains the corresponding name as localized for a particular display locale.

Special Considerations

This function can move memory in Carbon and Mac OS 9.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.
Available in Mac OS X 10.0 and later.

Declared In
`MacLocales.h`

LocaleOperationGetLocales

Obtains the list of available locale-and-variant combinations for a given class of operation.

```
OSStatus LocaleOperationGetLocales (
   LocaleOperationClass opClass,
   ItemCount maxLocaleCount,
   ItemCount *actualLocaleCount,
   LocaleAndVariant localeVariantList[]
);
```

Parameters

opClass
A `LocaleOperationClass` value identifying the operation class for which you are requesting availability information.

maxLocaleCount

An `ItemCount` value specifying the size of the array passed in the `localeVariantList` parameter. To determine how much memory to allocate for the list, you can call the function `LocaleOperationCountLocales` (page 1261). `LocaleOperationCountLocales` produces a count of the locale-and-variant combinations available for a given operation class in its `localeCount` parameter.

actualLocaleCount

A pointer to an `ItemCount` value. On return, the value contains the actual length of the list produced in the `localeVariantList` parameter.

localeVariantList

An array of `LocaleAndVariant` values. On return, the array contains a list of the available combinations of locale and operation variant for a specified operation class. The `LocaleAndVariant.opVariant` field may be 0 for entries that do not have a specific operation variant.

Return Value

A result code. If the value specified in the `maxLocaleCount` parameter is too small for the complete list, `LocaleOperationGetLocales` returns `kLocalesBufferTooSmallErr` (-30001). The function returns `paramErr` if `opClass` is 0 or if either the `actualLocaleCount` or `localeVariantList` parameter is `NULL`.

Discussion

The `LocaleOperationGetLocales` function provides a list of all the combinations of locales and operation variants that are available for a given class of operations, such as collation.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.

Available in Mac OS X 10.0 and later.

Declared In

`MacLocales.h`

LocaleOperationGetName

Obtains the localized name for an operation class, based on the requested language for the localized name.

```
OSStatus LocaleOperationGetName (
   LocaleOperationClass opClass,
   LocaleRef displayLocale,
   UniCharCount maxNameLen,
   UniCharCount *actualNameLen,
   UniChar displayName[]
);
```

Parameters

opClass

A `LocaleOperationClass` value identifying the operation class for which you are requesting a localized name.

displayLocale

A `LocaleRef` value indicating the requested language for the name produced in the `displayName` parameter. If `LocaleOperationGetName` cannot find a name in the requested language, it produces a name in the default display locale.

maxNameLen

A UniCharCount value specifying the length of the UniChar array being passed in the displayName parameter. An array of 64 characters is typically adequate.

actualNameLen

A pointer to a UniCharCount value. On return, the value contains the actual length of the name produced in the displayName parameter.

displayName

An array of UniChar values. On return, the array contains the requested name as localized for a particular display locale.

Return Value

A result code. If LocaleOperationGetName cannot find a name that matches the language specified in the displayLocale parameter, it returns kLocalesDefaultDisplayStatus (-30029). If the value specified in the maxNameLen parameter is too small for the requested string, LocaleOperationGetName returns kLocalesBufferTooSmallErr (-30001).

Discussion

For a particular operation class, the LocaleOperationGetName function obtains the corresponding name as localized for a particular display locale. Alternately, you can use repeated calls to the function LocaleOperationGetIndName (page 1262) to iterate through all of the available display locales to create a list of the corresponding names as localized in each of the display locales.

Special Considerations

This function can move memory in Carbon and Mac OS 9.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.

Available in Mac OS X 10.0 and later.

Declared In

MacLocales.h

LocaleRefFromLangOrRegionCode

Converts a Mac OS language or region code to a locale reference.

```
OSStatus LocaleRefFromLangOrRegionCode (
   LangCode lang,
   RegionCode region,
   LocaleRef *locale
);
```

Parameters

lang

A Mac OS LangCode value, specifying the language for which to create a LocaleRef. If you wish to specify only a region, not a language as well, you can pass the constant kTextLanguageDontCare (from TextCommon.h). The LocaleRefFromLangOrRegionCode function requires values for either the lang or region parameters, or both.

region

A Mac OS `RegionCode` value, specifying the region for which to create a `LocaleRef`. If you wish to specify only a language, not a region as well, you can pass the constant `kTextRegionDontCare` (from TextCommon.h). The `LocaleRefFromLangOrRegionCode` function requires values for either the `lang` or `region` parameters, or both.

locale

A pointer to a value of type `LocaleRef`. On return, the `LocaleRef` value contains a valid reference to locale data. Note that the `LocaleRefFromLangOrRegionCode` function can move memory, so the `locale` parameter should not point to memory that can move.

Return Value

A result code. The `LocaleRefFromLangOrRegionCode` function returns `paramErr` if no value is provided for either the `lang` or `region` parameters, or if both values are provided but they are inconsistent, or if a value provided is invalid. The function returns `paramErr` if the `locale` parameter is `NULL`. It can also return memory errors or resource errors. Finally, if the tables used for mapping language and region codes are invalid, the function returns `kLocalesTableFormatErr` (-30002) in CarbonLib and Mac OS 9 (in Mac OS 8.6, `paramErr` is returned).

Discussion

A `LocaleRef` is an opaque type that references static locale data. You typically provide a `LocaleRef` value to the locale-sensitive Unicode Utilities functions and to some Locale Utilities functions such as `LocaleGetName` (page 1258) and `LocaleOperationGetName` (page 1264).

You can use the `LocaleRefFromLangOrRegionCode` function to convert a Mac OS language and/or region code to a `LocaleRef` value. Additionally, you can use the function `LocaleRefFromLocaleString` (page 1266) to convert a locale part string—or an Internet language tag or POSIX/Java locale string—to a `LocaleRef` value.

You should not save `LocaleRef` values in a file or other persistent storage, because a given `LocaleRef` is not guaranteed to be valid across multiple launches of your application. For persistent storage, you can either save the original value or string used to construct the `LocaleRef`, or you can use the function `LocaleRefGetPartString` (page 1267) to convert a `LocaleRef` to a locale part string and save that.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.

Available in Mac OS X 10.0 and later.

Declared In

`MacLocales.h`

LocaleRefFromLocaleString

Converts a string containing locale data to a locale reference.

```
OSStatus LocaleRefFromLocaleString (
   const char localeString[],
   LocaleRef *locale
);
```

Parameters

localeString

An ASCII string containing an Internet RFC 1766-style language tag, or a POSIX-style or Java-style locale string, or a Mac OS locale part string. A locale part string is an ASCII string whose full form is "lan-var.sc-v_rg-v", where "lan-var" specifies the language and variant, ".sc-v" specifies the script & variant, and "_rg-v" specifies the region and variant. Any of those three parts can be omitted furthermore, the variant part "-var" or "-v" within any of those parts may be omitted.

locale

A pointer to a value of type `LocaleRef`. On return, the `LocaleRef` value contains a valid reference to locale data. Note that the `LocaleRefFromLocaleString` function can move memory, so the `locale` parameter should not point to memory that can move.

Return Value

A result code. The function returns `paramErr` if the `localeString` or `locale` parameters are `NULL`, or if the value provided in `localeString` is invalid. It can also return memory errors.

Discussion

A `LocaleRef` is an opaque type that references static locale data. You typically provide a `LocaleRef` value to the locale-sensitive Unicode Utilities functions and to some Locale Utilities functions such as `LocaleGetName` (page 1258) and `LocaleOperationGetName` (page 1264).

You can use the `LocaleRefFromLocaleString` function to convert a locale part string—or an Internet language tag or POSIX/Java locale string—to a `LocaleRef` value. Additionally, you can use the function `LocaleRefFromLangOrRegionCode` (page 1265) to convert a Mac OS language and/or region code to a `LocaleRef` value.

You should not save `LocaleRef` values in a file or other persistent storage, because a given `LocaleRef` is not guaranteed to be valid across multiple launches of your application. For persistent storage, you can either save the original value or string used to construct the `LocaleRef`, or you can use the function `LocaleRefGetPartString` (page 1267) to convert a `LocaleRef` back to a locale part string and save that.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.

Available in Mac OS X 10.0 and later.

Declared In

`MacLocales.h`

LocaleRefGetPartString

Converts a locale reference to a string containing locale data.

```
OSStatus LocaleRefGetPartString (
   LocaleRef locale,
   LocalePartMask partMask,
   ByteCount maxStringLen,
   char partString[]
);
```

Parameters

locale

The `LocaleRef` value to convert to string format. You can pass `NULL` for the default system locale.

partMask

A `LocalePartMask` value specifying the kinds of locale information to include in the string. Available kinds of locale information include language, script, region, and their respective variants.

maxStringLen

A `ByteCount` value specifying the length of the `char` array being passed in the `partString` parameter. The amount of storage should typically be at least 4 times the number of parts requested.

partString

An array of `char` values. On return, the array contains the new part string. The full form of the returned string is "lan-var.sc-v_rg-v" where "lan-var" specifies the language and variant, ".sc-v" specifies the script & variant, and "_rg-v" specifies the region and variant. Fields not specified by the `partMask` parameter (as well as any field separator that precedes them) are not included in the returned string.

Return Value

A result code. The function returns `paramErr` if the `partString` parameter is `NULL`, or if the locale is invalid. It returns `kLocalesBufferTooSmallErr` (-30001) if the value supplied in the `maxStringLen` parameter is too small for the requested string.

Discussion

The `LocaleRefGetPartString` function converts a `LocaleRef` value to a string containing locale information. You can use a locale part string to tag or specify language or locale in persistent storage.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.

Available in Mac OS X 10.0 and later.

Declared In

`MacLocales.h`

LocaleStringToLangAndRegionCodes

Converts a string containing locale data to Mac OS language and region codes.

```
OSStatus LocaleStringToLangAndRegionCodes (
   const char localeString[],
   LangCode *lang,
   RegionCode *region
);
```

Parameters

localeString

An ASCII string containing an Internet RFC 1766-style language tag, or a POSIX-style or Java-style locale string, or a Mac OS locale part string. If you have a `LocaleRef` value, you can obtain a locale string from the function `LocaleRefGetPartString` (page 1267). A locale part string is an ASCII string whose full form is "lan-var.sc-v_rg-v", where "lan-var" specifies the language and variant, ".sc-v" specifies the script & variant, and "_rg-v" specifies the region and variant. Any of those three parts can be omitted furthermore, the variant part "-var" or "-v" within any of those parts may be omitted.

lang

A pointer to a `LangCode` value, or pass `NULL`, if you do not want to obtain a language code. On return, `LocaleStringToLangAndRegionCodes` produces the appropriate Mac OS language code for the locale part string provided in the `localeString` parameter. Note that you can set either the `lang` or the `region` parameter to `NULL`, but not both.

region

A pointer to a `RegionCode` value or pass `NULL`, if you do not want to obtain a region code. On return, `LocaleStringToLangAndRegionCodes` produces the appropriate Mac OS region code for the locale part string provided in the `localeString` parameter. Note that you can set either the `region` or the `lang` parameter to `NULL`, but not both.

Return Value

A result code. If the `localeString` parameter is `NULL` or if the string cannot be mapped to an existing language code and region code, the `LocaleStringToLangAndRegionCodes` function returns `paramErr`. If the required resource is invalid, the function can return `kLocalesTableFormatErr` (-30002). The function can also return Resource Manager errors.

Discussion

The `LocaleStringToLangAndRegionCodes` function maps from a locale string to a combination of Mac OS language code and region code.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Declared In

`MacLocales.h`

Data Types

LocaleAndVariant

Identifies a specific locale and/or sub-class of locale-sensitive operations.

```
struct LocaleAndVariant {
    LocaleRef locale;
    LocaleOperationVariant opVariant;
};
typedef struct LocaleAndVariant LocaleAndVariant;
```

Fields

`locale`

A `LocaleRef` value, encapsulating information regarding language, script, and/or region.

`opVariant`

A `LocaleOperationVariant` value, identifying an individual sub-class of locale-sensitive operations. The `opVariant` field of a `LocaleAndVariant` value may be 0 for entries that do not have a specific operation variant.

Discussion

You use the function `LocaleOperationGetLocales` (page 1263) to obtain a list of all the locale-and-variant combinations available for a given class of operations. The `LocaleOperationGetLocales` function specifies these combinations using the `LocaleAndVariant` type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacLocales.h`

LocaleNameMask

Specifies named parts of locale data.

```
typedef UInt32 LocaleNameMask;
```

Discussion

The bits set in a `LocaleNameMask` value determine the kind of localized names that are produced by the functions `LocaleGetName` (page 1258) and `LocaleGetIndName` (page 1257). For a description of the `LocaleNameMask` values, see "Locale Name Masks" (page 1272).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacLocales.h`

LocaleOperationClass

Identifies individual classes of locale-sensitive operations.

```
typedef FourCharCode LocaleOperationClass;
```

Discussion

Locale-sensitive Unicode Utilities operations fall into several classes, such as collation, text break determination, and date and time formatting. You use the `LocaleOperationClass` type to specify these classes.

Specific `LocaleOperationClass` values depend on the Locale Utilities client. Values from Unicode Utilities include `kUnicodeCollationClass ('ucol')` and `kUnicodeTextBreakClass ('ubrk')`. For example, the locales and collation variants available for collation operations can be determined by calling the functions `LocaleOperationCountLocales` (page 1261) and `LocaleOperationGetLocales` (page 1263) with the `opClass` parameter set to the `kUnicodeCollationClass` constant.

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

Declared In
`MacLocales.h`

LocaleOperationVariant

Identifies individual sub-classes of locale-sensitive operations.

```
typedef FourCharCode LocaleOperationVariant;
```

Discussion
Locale-sensitive Unicode Utilities operations fall into several classes, such as collation, text break determination, and date and time formatting. You use the type `LocaleOperationClass` (page 1270) to specify these classes. Some of these classes also have sub-classes of their operations. For example, sub-classes of collation operations include dictionary vs. bibliographic collation, and radical-stroke vs. pinyin collation.

For classes such as collation, the `LocaleOperationVariant` type provides an operation-specific variant field. This field is analogous to the `@variant` part of a POSIX locale string or the final `_VARIANT` part of a Java locale string.

You can use `LocaleOperationVariant` values in the functions `LocaleGetName` (page 1258) , `LocaleGetIndName` (page 1257) , and `LocaleCountNames` (page 1256) to obtain localized names for operation variants.

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

Declared In
`MacLocales.h`

LocalePartMask

Specifies kinds of locale data.

```
typedef UInt32 LocalePartMask;
```

Discussion
The bits set in a `LocalePartMask` value determine the kinds of locale information that are produced by the function `LocaleRefGetPartString` (page 1267). For a description of the `LocalePartMask` values, see "Locale Part Masks" (page 1273).

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

Declared In
MacLocales.h

LocaleRef

An opaque type that refers to locale information.

```
typedef struct OpaqueLocaleRef * LocaleRef;
```

Discussion
A LocaleRef is an opaque type that references static locale data. You typically provide a LocaleRef value to the locale-sensitive Unicode Utilities functions and to some Locale Utilities functions such as LocaleGetName (page 1258) and LocaleOperationGetName (page 1264).

To obtain a LocaleRef, you can use the function LocaleRefFromLangOrRegionCode (page 1265) to convert a Mac OS language and/or region code to a LocaleRef value. Additionally, you can use the function LocaleRefFromLocaleString (page 1266) to convert a locale part string—or an Internet language tag or POSIX/Java locale string—to a LocaleRef value.

You should not save LocaleRef values in a file or other persistent storage, since a given LocaleRef is not guaranteed to be valid across multiple launches of your application. For persistent storage, you can either save the original value or string used to construct the LocaleRef, or you can use the function LocaleRefGetPartString (page 1267) to convert a LocaleRef to a locale part string and save that.

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

Declared In
MacLocales.h

Constants

Locale Name Masks

Specify named parts of locale data.

```
enum {
    kLocaleNameMask = 1L << 0,
    kLocaleOperationVariantNameMask = 1L << 1,
    kLocaleAndVariantNameMask = 0x00000003
};
```

Constants
kLocaleNameMask
If the bit specified by this mask is set, then the name for a locale is specified.
Available in Mac OS X v10.0 and later.
Declared in MacLocales.h.

`kLocaleOperationVariantNameMask`
If the bit specified by this mask is set, then the name for an operation variant is specified.

Available in Mac OS X v10.0 and later.

Declared in `MacLocales.h`.

`kLocaleAndVariantNameMask`
If the bits specified by this mask are set, then the name for a locale-and-variant combination is specified.

Available in Mac OS X v10.0 and later.

Declared in `MacLocales.h`.

Discussion
The bits set in a `LocaleNameMask` value determine the kind of localized names that are produced by the functions `LocaleGetName` (page 1258) and `LocaleGetIndName` (page 1257). With a `LocaleNameMask` value, you can specify the name of a locale alone, the name of an operation variant alone, or the name for the combination of the two.

Locale Part Masks

Specify kinds of locale data.

```
enum {
    kLocaleLanguageMask = 1L << 0,
    kLocaleLanguageVariantMask = 1L << 1,
    kLocaleScriptMask = 1L << 2,
    kLocaleScriptVariantMask = 1L << 3,
    kLocaleRegionMask = 1L << 4,
    kLocaleRegionVariantMask = 1L << 5,
    kLocaleAllPartsMask = 0x0000003F
};
```

Constants
`kLocaleLanguageMask`
If the bit specified by this mask is set, then a ISO 639-1 or -2 language code is specified.

Available in Mac OS X v10.0 and later.

Declared in `MacLocales.h`.

`kLocaleLanguageVariantMask`
If the bit specified by this mask is set, then a custom string for a language variant is specified.

Available in Mac OS X v10.0 and later.

Declared in `MacLocales.h`.

`kLocaleScriptMask`
If the bit specified by this mask is set, then a ISO 15924 script code is specified.

Available in Mac OS X v10.0 and later.

Declared in `MacLocales.h`.

`kLocaleScriptVariantMask`
If the bit specified by this mask is set, then a custom string for a script variant is specified.

Available in Mac OS X v10.0 and later.

Declared in `MacLocales.h`.

`kLocaleRegionMask`
If the bit specified by this mask is set, then a ISO 3166 country/region code is specified.

Available in Mac OS X v10.0 and later.

Declared in `MacLocales.h`.

`kLocaleRegionVariantMask`
If the bit specified by this mask is set, then a custom string for a region variant is specified.

Available in Mac OS X v10.0 and later.

Declared in `MacLocales.h`.

`kLocaleAllPartsMask`
If the bits specified by this mask are set, then all types (language and variant, script and variant, and region and variant) of locale information are specified.

Available in Mac OS X v10.0 and later.

Declared in `MacLocales.h`.

Discussion
The bits set in a `LocalePartMask` value determine the kinds of locale information that are produced by the function `LocaleRefGetPartString` (page 1267).

Mathematical and Logical Utilities Reference

Framework:	CoreServices/CoreServices.h
Declared in	ToolUtils.h FixMath.h Math64.h fp.h MacTypes.h

Overview

Important: This is a preliminary document. Although it has been reviewed for technical accuracy, it is not final. Apple Computer is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. For information about updates to this and other developer documentation, you can check the ADC Reference Library Revision List. To receive notification of documentation updates, you can sign up for ADC's free Online Program and receive the weekly Apple Developer Connection News email newsletter. (See http://developer.apple.com/membership for more details about the Online Program.)

You can use the Mathematical and Logical Utilities to perform mathematical and logical operations in Mac OS X programming. This document describes functions you can use to:

- Perform low-level logical manipulation of bits and bytes when using a compiler that does not directly support such manipulations.
- Save disk space by using simple compression and decompression routines.
- Obtain a pseudorandom number.
- Perform mathematical operations with two fixed-point data types supported directly by the Operating System.
- Convert numeric variables of different types.

With the exception of the mathematical operations and conversions, these utilities are intended for programmers who occasionally need to access some of these features and do not require that the algorithms used to implement them be sophisticated. For example, if you are developing an advanced mathematical application, the pseudorandom number generator built into Mac OS might be too simplistic to fit your needs. Similarly, if you wish to access individual bits of memory in a time-critical loop, these routines are probably too slow to be practical.

Carbon supports the Mathematical and Logical Utilities, with the exception of those functions that are 68K-specific. However there are several important differences between the implementation of the Mathematical and Logical Utilities in Mac OS 9 and its implementation in Mac OS X.

The implementation in Carbon on Mac OS X of many floating-point functions defined in `fp.h` is not as accurate as the implementation of those functions in MathLib on Mac OS 8 and 9 (as accessed either directly or through CarbonLib). There are a number of reasons for this difference, including the different expectations of Mac OS 9 and UNIX floating-point clients, compiler limitations, and the need in for an implementation that's independent of assumptions about the size and layout of floating-point data types.

Functions which take parameters or return values of type long double are not exported by the Core Services framework on Mac OS X. Instead, these functions have been replaced with macros that map to the corresponding double-typed functions. While these functions are exported by CarbonLib, CFM applications calling these functions on Mac OS X should note that the implementations of the long double functions on Mac OS X actually have only double precision, with the following four exceptions: `num2decl`, `dec2numl`, `x80told`, and `ldtox80`.

Functions by Task

Converting Among 32-Bit Numeric Types

`Fix2Frac` (page 1297)
Converts a `Fixed` number to a `Fract` number.

`Fix2Long` (page 1298)
Converts a `Fixed` number to a `LongInt` number.

`Frac2Fix` (page 1305)
Converts a `Fract` number to a `Fixed` number.

`Long2Fix` (page 1314)
Converts a `LongInt` number to a `Fixed` number.

Converting Between Fixed-Point and Floating-Point Values

`FixedToFloat` (page 1300)
Converts a `Fixed` number to a `float` number.

`FractToFloat` (page 1308)
Converts a `Fract` number to a `float` number.

`FloatToFixed` (page 1302)
Converts a `float` number to a `Fixed` number.

`FloatToFract` (page 1303)
Converts a `float` number to a `Fract` number.

`Fix2X` (page 1298)
Converts a `Fixed` number to an `Extended` number.

`Frac2X` (page 1305)
Converts a `Fract` number to an `Extended` number.

`X2Fix` (page 1344)
Converts an `Extended` number to a `Fixed` number.

`X2Frac` (page 1344)
Converts an `Extended` number to a `Fract` number.

Converting Between Fixed-Point and Integral Values

`FixRatio` (page 1301)
Obtains the `Fixed` equivalent of a fraction.

`FixRound` (page 1302)
Rounds a fixed-point number to the nearest integer.

Getting and Setting Memory Values

`HiWord` (page 1310)
Obtains the high-order word of a long word.

`LoWord` (page 1315)
Obtains the low-order word of a long word.

Multiplying and Dividing Fixed-Point Numbers

`FixDiv` (page 1299)
Divides two variables of the same type (`Fixed`, `Fract`, or `LongInt`) or to divide a `LongInt` or `Fract` number by a `Fixed` number.

`FixMul` (page 1300)
Multiplies a variable of type `Fixed` with another variable of type `Fixed` or with a variable of type `Fract` or `LongInt`.

`FracDiv` (page 1306)
Divides two variables of the same type (`Fract`, `Fixed`, or `LongInt`) or to divide a `LongInt` or `Fixed` number by a `Fract` number.

`FracMul` (page 1307)
Multiplies a variable of type `Fract` with another variable of type `Fract` or with a variable of type `Fixed` or `LongInt`.

Performing Calculations on Fixed-Point Numbers

`FixATan2` (page 1299)
Obtains a fast approximation of the arctangent of a fraction.

`FracCos` (page 1306)
Obtains a fast approximation of the cosine of a `Fixed` number.

`FracSin` (page 1307)
Obtains a fast approximation of the sine of a `Fixed` number.

FracSqrt (page 1308)
Obtains the square root of a Fract number.

Performing Logical Operations

BitAnd (page 1287)
Performs the AND logical operation on two long words.

BitNot (page 1288)
Performs the NOT logical operation on a long word.

BitOr (page 1288)
Performs the OR logical operation on two long words.

BitShift (page 1289)
Shifts bits in a long word.

BitXor (page 1290)
Performs the XOR logical operation on two long words.

Testing and Setting Bits

BitClr (page 1287)
Clears a particular bit (to a value of 0).

BitSet (page 1289)
Sets a particular bit (to a value of 1).

BitTst (page 1290)
Determines whether a given bit is set.

Miscellaneous Functions

acos (page 1284)

acosh (page 1285)

annuity (page 1285)

asin (page 1285)

asinh (page 1286)

atan (page 1286)

atan2 (page 1286)

atanh (page 1287)

Functions

acos

```
double_t acos (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

acosh

```
double_t acosh (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

annuity

```
double annuity (
   double rate,
   double periods
);
```

Parameters

rate

periods

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

asin

```
double_t asin (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

asinh

```
double_t asinh (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

atan

```
double_t atan (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

atan2

```
double_t atan2 (
   double_t y,
   double_t x
);
```

Parameters

y

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

atanh

```
double_t atanh (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

BitAnd

Performs the AND logical operation on two long words.

```
long BitAnd (
   long value1,
   long value2
);
```

Parameters

value1
A long word.

value2
A long word.

Return Value
A long word that is the result of the AND operation on the long words passed as arguments. Each bit in the returned value is set if and only if the corresponding bit is set in both value1 and value2.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
ToolUtils.h

BitClr

Clears a particular bit (to a value of 0).

```
void BitClr (
   void *bytePtr,
   long bitNum
);
```

Parameters

bytePtr
A pointer to a byte in memory.

bitNum
The bit to be cleared, specified as a positive offset from the high-order bit of the byte pointed to by the `bytePtr` parameter. The bit being cleared need not be in the same byte pointed to by `bytePtr`.

Special Considerations

The bit numbering scheme used by the `BitClr` function is the opposite of the MC680x0 numbering. To convert an MC680x0 bit number to the format required by the `BitClr` function, subtract the MC680x0 bit number from the highest bit number.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`ToolUtils.h`

BitNot

Performs the `NOT` logical operation on a long word.

```
long BitNot (
   long value
);
```

Parameters

value
A long word.

Return Value
A long word that is the result of the `NOT` operation on the long word passed in as an argument. Each bit in the returned value is set if and only if the corresponding bit is not set in `value`.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`ToolUtils.h`

BitOr

Performs the `OR` logical operation on two long words.

```
long BitOr (
   long value1,
   long value2
);
```

Parameters

value1
A long word.

value2
A long word.

Return Value
A long word that is the result of the `OR` operation on the long words passed as arguments. Each bit in the returned value is set if and only if the corresponding bit is set in `value1` or `value2`, or in both `value1` and `value2`.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`ToolUtils.h`

BitSet

Sets a particular bit (to a value of 1).

```
void BitSet (
   void *bytePtr,
   long bitNum
);
```

Parameters

`bytePtr`
A pointer to a byte in memory.

`bitNum`
The bit to be set, specified as a positive offset from the high-order bit of the byte pointed to by the `bytePtr` parameter. The bit being set need not be in the byte pointed to by `bytePtr`.

Special Considerations

The bit numbering scheme used by the `BitSet` function is the opposite of the MC680x0 numbering. To convert an MC680x0 bit number to the format required by the `BitSet` function, subtract the MC680x0 bit number from the highest bit number.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`ToolUtils.h`

BitShift

Shifts bits in a long word.

```
long BitShift (
   long value,
   short count
);
```

Parameters

`value`
A long word.

count

The number of bits to shift. If this number is positive, `BitShift` shifts this many positions to the left; if this number is negative, `BitShift` shifts this many positions to the right. The value in this parameter is converted to the result of `MOD` 32.

Return Value
A long word that is the result of shifting the bits in the long word passed in as an argument. The shift's direction and extent are determined by the `count` parameter. Zeroes are shifted into empty positions regardless of the direction of the shift.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`ToolUtils.h`

BitTst

Determines whether a given bit is set.

```
Boolean BitTst (
   const void *bytePtr,
   long bitNum
);
```

Parameters

bytePtr

A pointer to a byte in memory.

bitNum

The bit to be tested, specified as a positive offset from the high-order bit of the byte pointed to by the `bytePtr` parameter. The bit being tested need not be in the byte pointed to by `bytePtr`.

Return Value
`TRUE` if the specified bit is set (that is, has a value of 1) and `FALSE` if the bit is cleared (that is, has a value of 0).

Special Considerations

The bit numbering scheme used by the `BitTst` function is the opposite of the MC680x0 numbering. To convert an MC680x0 bit number to the format required by the `BitTst` function, subtract the MC680x0 bit number from the highest bit number.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`ToolUtils.h`

BitXor

Performs the `XOR` logical operation on two long words.

```
long BitXor (
   long value1,
   long value2
);
```

Parameters

value1

A long word.

value2

A long word.

Return Value

A long word that is the result of the XOR operation on the long words passed in as arguments. Each bit in the returned value is set if and only if the corresponding bit is set in either value1 or value2, but not in both value1 and value2.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

ToolUtils.h

ceil

```
double_t ceil (
   double_t x
);
```

Parameters

x

Return Value

Availability

Available in Mac OS X version 10.0 and later.

Related Sample Code

Aperture Edit Plugin - Borders & Titles

Declared In

fp.h

compound

```
double compound (
   double rate,
   double periods
);
```

Parameters

rate

periods

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

copysign

```
double_t copysign (
   double_t x,
   double_t y
);
```

Parameters

x

y

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

cos

```
double_t cos (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

cosh

```
double_t cosh (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

dec2f

```
float dec2f (
   const decimal *d
);
```

Parameters

d

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

dec2l

```
long dec2l (
   const decimal *d
);
```

Parameters

d

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

dec2num

```
double_t dec2num (
   const decimal *d
);
```

Parameters

d

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

dec2s

```
short dec2s (
   const decimal *d
);
```

Parameters

d

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

dec2str

```
void dec2str (
   const decform *f,
   const decimal *d,
   char *s
);
```

Parameters

f

d

s

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

dtox80

```
void dtox80 (
   const double *x,
   extended80 *x80
);
```

Parameters

x

x80

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

erf

```
double_t erf (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

erfc

```
double_t erfc (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

exp

```
double_t exp (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

exp2

```
double_t exp2 (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

expm1

```
double_t expm1 (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

fabs

```
double_t fabs (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
HID Calibrator
HID Config Save
HID Explorer
SIMD Primer

Declared In
`fp.h`

fdim

```
double_t fdim (
   double_t x,
   double_t y
);
```

Parameters

x

y

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

Fix2Frac

Converts a `Fixed` number to a `Fract` number.

```
Fract Fix2Frac (
   Fixed x
);
```

Parameters

x

The `Fixed` number to be converted to a `Fract` number.

Return Value
The `Fract` number equivalent to the `Fixed` number `x`. If `x` is greater than the maximum representable `Fract` number, the `Fix2Frac` function returns $7FFFFFFF. If `x` is less than the negative number with the highest absolute value, `Fix2Frac` returns $80000000.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`FixMath.h`

Fix2Long

Converts a `Fixed` number to a `LongInt` number.

```
SInt32 Fix2Long (
   Fixed x
);
```

Parameters

x
The `Fixed` number to be converted to a long integer.

Return Value
The long integer nearest to the `Fixed` number `x`. If `x` is halfway between two integers (0.5), it is rounded to the integer with the higher absolute value.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`FixMath.h`

Fix2X

Converts a `Fixed` number to an `Extended` number.

```
double Fix2X (
   Fixed x
);
```

Parameters

x
The `Fixed` number to be converted to an `Extended` number.

Return Value
The `Extended` equivalent of the `Fixed` number `x`.

Special Considerations

`Fix2X` does not move memory; you can call it at interrupt time.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
FixMath.h

FixATan2

Obtains a fast approximation of the arctangent of a fraction.

```
Fixed FixATan2 (
   SInt32 x,
   SInt32 y
);
```

Parameters

x

The numerator of the fraction whose arctangent is to be obtained. This variable can be a `LongInt`, `Fixed`, or `Fract` number.

y

The denominator of the fraction whose arctangent is to be obtained. The number supplied in this variable must be of the same type as that of the number supplied in the `x` parameter.

Return Value
The arctangent of `y/x`, in radians.

Discussion
The approximation of p/4 used to compute the arctangent is the hexadecimal value 0.C910, making the approximation of p equal to 3.1416015625, while p itself equals 3.14159265.... Thus `FixATan2(1, 1)` equals the equivalent of the hexadecimal value 0.C910. Despite the approximation of p, the arctangent value obtained will usually be correct to several decimal places.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
FixMath.h

FixDiv

Divides two variables of the same type (`Fixed`, `Fract`, or `LongInt`) or to divide a `LongInt` or `Fract` number by a `Fixed` number.

```
Fixed FixDiv (
   Fixed x,
   Fixed y
);
```

Parameters

x

The first operand, which can be a variable of type `Fixed` or a variable of type `Fract` or `LongInt`.

y

The second operand, which can be a variable of type `Fixed` or it can be a variable of the same type as the variable in parameter `x`.

Return Value
The quotient of the numbers in `x` and `y`. If the `y` parameter is in the format of a `Fixed` number, then the `x` parameter can be in the format of a `Fixed`, `Fract`, or `LongInt` number. If the `y` parameter is in the format of a `Fract` or `LongInt` number, then the `x` parameter must be in the same format.

The returned value is in the format of a `Fixed` number if both `x` and `y` are both `Fixed` numbers, both `Fract` numbers, or both `LongInt` numbers. Otherwise, the returned value is the same type as the number in the `x` parameter.

Division by zero results in $8000000 if `x` is negative, and $7FFFFFFF otherwise; thus the special case 0/0 yields $7FFFFFFF.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
SoftVDigX

Declared In
`FixMath.h`

FixedToFloat

Converts a `Fixed` number to a `float` number.

```
float FixedToFloat (
    Fixed x
);
```

Parameters

x
The `Fixed` number to be converted.

Return Value
The `float` equivalent of the `Fixed` number.

Discussion
This function is implemented as an inline macro.

Availability
Available in Mac OS X version 10.3 and later.

Declared In
`FixMath.h`

FixMul

Multiplies a variable of type `Fixed` with another variable of type `Fixed` or with a variable of type `Fract` or `LongInt`.

```
Fixed FixMul (
   Fixed a,
   Fixed b
);
```

Parameters

a

The first operand, which can be a variable of type `Fixed` or a variable of type `Fract` or `LongInt`.

b

The second operand, which can be a variable of type `Fixed` or a variable of type `Fract` or `LongInt`.

Return Value
The product of the numbers in `a` and `b`. At least one of `a` and `b` should be a variable of type `Fixed`.

The returned value is in the format of a `LongInt` if one of `a` or `b` is a `LongInt`. It is a `Fract` number if one of `a` or `b` is `Fract`. It is a `Fixed` number if both `a` and `b` are `Fixed` numbers.

Overflows are set to the maximum representable value with the correct sign ($80000000 for negative results and $7FFFFFFF for positive results).

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`FixMath.h`

FixRatio

Obtains the `Fixed` equivalent of a fraction.

```
Fixed FixRatio (
   short numer,
   short denom
);
```

Parameters

numer

The numerator of the fraction.

denom

The denominator of the fraction.

Return Value
The `Fixed` equivalent of the fraction `numer/denom`.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
SoftVDigX

Declared In
`FixMath.h`

FixRound

Rounds a fixed-point number to the nearest integer.

```
short FixRound (
   Fixed x
);
```

Parameters

x

The `Fixed` number to be rounded.

Return Value

The `Integer` number nearest the `Fixed` number `x`. If the value is halfway between two integers (0.5), it is rounded up. Thus, 4.5 is rounded to 5, and –3.5 is rounded to –3.

Discussion

To round a negative `Fixed` number so that values halfway between two integers are rounded to the number with the higher absolute value, negate the number, round it, and then negate it again.

Availability

Available in Mac OS X version 10.0 and later.

Related Sample Code

SoftVDigX

Declared In

`FixMath.h`

FloatToFixed

Converts a `float` number to a `Fixed` number.

```
Fixed FloatToFixed (
    float x
);
```

Parameters

x

The `float` number to be converted.

Return Value

The `Fixed` equivalent of the `float` number.

Discussion

This function is implemented as an inline macro.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`FixMath.h`

FloatToFract

Converts a `float` number to a `Fract` number.

```
Fract FloatToFract (
    float x
);
```

Parameters

x

The `float` number to be converted.

Return Value

The `Fract` equivalent of the `float` number.

Discussion

This function is implemented as an inline macro.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`FixMath.h`

floor

```
double_t floor (
   double_t x
);
```

Parameters

x

Return Value

Availability

Available in Mac OS X version 10.0 and later.

Related Sample Code

Aperture Edit Plugin - Borders & Titles

WhackedTV

Declared In

`fp.h`

fmax

```
double_t fmax (
   double_t x,
   double_t y
);
```

Parameters

x

y

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

fmin

```
double_t fmin (
   double_t x,
   double_t y
);
```

Parameters

x

y

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

fmod

```
double_t fmod (
   double_t x,
   double_t y
);
```

Parameters

x

y

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

fpclassify

```
long fpclassify (
    float x
);
```

Parameters

x

A value of type `float` or `double`.

Return Value
Returns one of the FP_ values. See `FP_SNAN` (page 1351).

Discussion
This function is implemented as an inline macro.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

Frac2Fix

Converts a `Fract` number to a `Fixed` number.

```
Fixed Frac2Fix (
   Fract x
);
```

Parameters

x

The `Fract` number to be converted to a `Fixed` number.

Return Value
The `Fixed` number that best approximates the `Fract` number `x`.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`FixMath.h`

Frac2X

Converts a `Fract` number to an `Extended` number.

```
double Frac2X (
   Fract x
);
```

Parameters

x

The `Fract` number to be converted to an `Extended` number.

Return Value
The `Extended` equivalent of the `Fract` number `x`.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`FixMath.h`

FracCos

Obtains a fast approximation of the cosine of a `Fixed` number.

```
Fract FracCos (
   Fixed x
);
```

Parameters

x

The `Fixed` number expressed in radians, whose cosine is to be calculated.

Return Value
The cosine, expressed in radians, of the `Fixed` number `x`.

Discussion
The approximation of p/4 used to compute the cosine is the hexadecimal value 0.C910, making the approximation of p equal to 3.1416015625, while p itself equals 3.14159265.... Despite the approximation of p, the cosine value obtained is usually correct to several decimal places.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`FixMath.h`

FracDiv

Divides two variables of the same type (`Fract`, `Fixed`, or `LongInt`) or to divide a `LongInt` or `Fixed` number by a `Fract` number.

```
Fract FracDiv (
   Fract x,
   Fract y
);
```

Parameters

x

The first operand, which can be a variable of type `Fract` or a variable of type `Fixed` or `LongInt`.

y

The second operand, which can be a variable of type `Fract` or a variable of the same type as the variable in parameter `a`.

Return Value
The quotient of the numbers in `a` and `b`. If the `b` parameter is in the format of a `Fract` number, then the `a` parameter can be in the format of a `Fract`, a `Fixed`, or a `LongInt` number. If the `b` parameter is in the format of a `Fixed` or a `LongInt` number, then the `a` parameter must be in the same format.

The returned value is in the format of a `Fract` number if `a` and `b` are both `Fract` numbers, both `Fixed` numbers, or both `LongInt` numbers. Otherwise, the returned value is in the same format as the number in the `a` parameter.

Division by zero results in $8000000 if `a` is negative, and $7FFFFFFF otherwise; thus the special case 0/0 yields $7FFFFFFF.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`FixMath.h`

FracMul

Multiplies a variable of type `Fract` with another variable of type `Fract` or with a variable of type `Fixed` or `LongInt`.

```
Fract FracMul (
   Fract x,
   Fract y
);
```

Parameters

x
The first operand, which can be a variable of type `Fract` or a variable of type `Fixed` or `LongInt`.

y
The second operand, which can be a variable of type `Fract` or a variable of type `Fixed` or `LongInt`.

Return Value
The product of the numbers in `a` and `b`. At least one of `a` or `b` should be a variable of type `Fract`.

The returned value is in the format of a `LongInt` number if one of `a` and `b` is a `LongInt` number. It is a `Fixed` number if one of `a` or `b` is a `Fixed` number. It is a `Fract` number if both `a` and `b` are `Fract` numbers.

Overflows are set to the maximum representable value with the correct sign ($80000000 for negative results and $7FFFFFFF for positive results).

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`FixMath.h`

FracSin

Obtains a fast approximation of the sine of a `Fixed` number.

```
Fract FracSin (
   Fixed x
);
```

Parameters

x

The `Fixed` number expressed in radians, whose sine is to be calculated.

Return Value
The sine, expressed in radians, of the `Fixed` number `x`.

Discussion
The approximation of p/4 used to compute the sine is the hexadecimal value 0.C910, making the approximation of p equal to 3.1416015625, while p itself equals 3.14159265.... Despite the approximation of p, the sine value obtained is usually correct to several decimal places.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`FixMath.h`

FracSqrt

Obtains the square root of a `Fract` number.

```
Fract FracSqrt (
   Fract x
);
```

Parameters

x

The `Fract` number to obtain a square root of. This parameter is interpreted as being unsigned in the range 0 through 4 – 2–30, inclusive. That is, the bit of the `Fract` number that ordinarily has weight -2 is instead interpreted as having weight 2.

Return Value
The square root of the specified `Fract` number. The result is unsigned in the range 0 through 2, inclusive.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`FixMath.h`

FractToFloat

Converts a `Fract` number to a `float` number.

```
float FixedToFract (
    Fract x
);
```

Parameters

x

The `Fract` number to be converted.

Return Value

The `float` equivalent of the `Fract` number.

Discussion

This function is implemented as an inline macro.

Availability

Available in Mac OS X version 10.3 and later.

Declared In

`FixMath.h`

frexp

```
double_t frexp (
   double_t x,
   int *exponent
);
```

Parameters

x

exponent

Return Value

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`fp.h`

gamma

```
double_t gamma (
   double_t x
);
```

Parameters

x

Return Value

Availability

Available in Mac OS X version 10.0 and later.

Related Sample Code

Gamma Filter for FxPlug and AE

SoftVDigX

Declared In
fp.h

HiWord

Obtains the high-order word of a long word.

```
SInt16 HiWord (
    SInt32 x
);
```

Parameters

x
The long word whose high word is to be returned.

Return Value
The high-order word of the long word specified by the `x` parameter.

Discussion
One use of this function is to obtain the integral part of a fixed-point number.

To copy a range of bytes from one memory location to another, you should ordinarily use the Memory Manager function, `BlockMove`.

Availability

Declared In
ToolUtils.h

hypot

```
double_t hypot (
   double_t x,
   double_t y
);
```

Parameters

x

y

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
pyport.h

isfinite

```
long isfinite (
    float x
);
```

Parameters

x

A value of type `float` or `double`.

Return Value
Returns a non-zero value only if the argument is finite.

Discussion
This function is implemented as an inline macro.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

isnan

```
long isnan (
    float x
);
```

Parameters

x

A value of type `float` or `double`.

Return Value
Returns a non-zero value only if the argument is not a number (NaN).

Discussion
This function is implemented as an inline macro.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

isnormal

```
long isnormal (
    float x
);
```

Parameters

x

A value of type `float` or `double`.

Return Value
Returns a non-zero value only if the argument is normalized.

Discussion
This function is implemented as an inline macro.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

ldexp

```
double_t ldexp (
   double_t x,
   int n
);
```

Parameters

x

n

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

lgamma

```
double_t lgamma (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

log

```
double_t log (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
dist_fft
FBOBunnies
FilterDemo
LSMSmartCategorizer
PBORenderToVertexArray

Declared In
`syslog.h`

log10

```
double_t log10 (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
WhackedTV

Declared In
`fp.h`

log1p

```
double_t log1p (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

log2

```
double_t log2 (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

logb

```
double_t logb (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

Long2Fix

Converts a `LongInt` number to a `Fixed` number.

```
Fixed Long2Fix (
   SInt32 x
);
```

Parameters

x

The long integer to be converted to a `Fixed` number.

Return Value
The `Fixed` number equivalent to the long integer `x`. If `x` is greater than the maximum representable fixed-point number, the `Long2Fix` function returns $7FFFFFFF. If `x` is less than the negative number with the highest absolute value, `Long2Fix` returns $80000000.

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
ASCIIMoviePlayerSample

Declared In
`FixMath.h`

LoWord

Obtains the low-order word of a long word.

```
SInt16 LoWord (
    SInt32 x
);
```

Parameters

x

The long word whose low word is to be returned.

Return Value
The low-order word of the long word specified by the `x` parameter.

Discussion
One use of this function is to obtain the fractional part of a fixed-point number.

To copy a range of bytes from one memory location to another, you should ordinarily use the Memory Manager function, `BlockMove`.

Availability

Declared In
`ToolUtils.h`

modf

```
double_t modf (
   double_t x,
   double_t *iptr
);
```

Parameters

x

iptr

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

modff

```
float modff (
   float x,
   float *iptrf
);
```

Parameters

x

iptrf

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

nan

```
double nan (
   const char *tagp
);
```

Parameters

tagp

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

nanf

```
float nanf (
   const char *tagp
);
```

Parameters

tagp

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

nearbyint

```
double_t nearbyint (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

nextafterd

```
double nextafterd (
   double x,
   double y
);
```

Parameters

x

y

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

nextafterf

```
float nextafterf (
   float x,
   float y
);
```

Parameters

x

y

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

num2dec

```
void num2dec (
   const decform *f,
   double_t x,
   decimal *d
);
```

Parameters

f

x

d

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

pi

```
pi ();
```

Parameters

Return Value

Availability

Declared In
`fp.h`

pow

```
double_t pow (
   double_t x,
   double_t y
);
```

Parameters

x

y

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
Gamma Filter for FxPlug and AE

WhackedTV

Declared In
`fp.h`

randomx

```
double_t randomx (
   double_t *x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

relation

```
relop relation (
   double_t x,
   double_t y
);
```

Parameters

x

y

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

remainder

```
double_t remainder (
   double_t x,
   double_t y
);
```

Parameters

x

y

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
SoftVDigX

Declared In
fp.h

remquo

```
double_t remquo (
   double_t x,
   double_t y,
   int *quo
);
```

Parameters

x

y

quo

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

rint

```
double_t rint (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

rinttol

```
long rinttol (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

round

```
double_t round (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

roundtol

```
long roundtol (
   double_t round
);
```

Parameters

round

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

S32Set

```
SInt32 S32Set (
   SInt64 value
);
```

Parameters

value

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64Absolute

```
SInt64 S64Absolute (
   SInt64 value
);
```

Parameters

value

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64Add

```
SInt64 S64Add (
   SInt64 left,
   SInt64 right
);
```

Parameters

x

y

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64And

```
Boolean S64And (
   SInt64 left,
   SInt64 right
);
```

Parameters

left

right

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64BitwiseAnd

```
SInt64 S64BitwiseAnd (
   SInt64 left,
   SInt64 right
);
```

Parameters

left

right

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64BitwiseEor

```
SInt64 S64BitwiseEor (
   SInt64 left,
   SInt64 right
);
```

Parameters

left

right

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64BitwiseNot

```
SInt64 S64BitwiseNot (
   SInt64 value
);
```

Parameters

value

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64BitwiseOr

```
SInt64 S64BitwiseOr (
   SInt64 left,
   SInt64 right
);
```

Parameters

left

right

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64Compare

```
SInt32 S64Compare (
   SInt64 left,
   SInt64 right
);
```

Parameters

left

right

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64Div

```
SInt64 S64Div (
   SInt64 dividend,
   SInt64 divisor
);
```

Parameters

dividend

divisor

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64Divide

```
SInt64 S64Divide (
   SInt64 dividend,
   SInt64 divisor,
   SInt64 *remainder
);
```

Parameters

dividend

divisor

remainder

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64Eor

```
Boolean S64Eor (
   SInt64 left,
   SInt64 right
);
```

Parameters

left

right

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64Max

```
SInt64 S64Max (
   void
);
```

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64Min

```
SInt64 S64Min (
   void
);
```

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64Multiply

```
SInt64 S64Multiply (
   SInt64 left,
   SInt64 right
);
```

Parameters

xparam

yparam

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64Negate

```
SInt64 S64Negate (
   SInt64 value
);
```

Parameters

value

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64Not

```
Boolean S64Not (
   SInt64 value
);
```

Parameters

value

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64Or

```
Boolean S64Or (
   SInt64 left,
   SInt64 right
);
```

Parameters

left

right

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64Set

```
SInt64 S64Set (
   SInt32 value
);
```

Parameters

value

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64SetU

```
SInt64 S64SetU (
   UInt32 value
);
```

Parameters

value

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64ShiftLeft

```
SInt64 S64ShiftLeft (
   SInt64 value,
   UInt32 shift
);
```

Parameters

value

shift

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64ShiftRight

```
SInt64 S64ShiftRight (
   SInt64 value,
   UInt32 shift
);
```

Parameters

value

shift

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

S64Subtract

```
SInt64 S64Subtract (
   SInt64 left,
   SInt64 right
);
```

Parameters

left

right

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

scalb

```
double_t scalb (
   double_t x,
   _scalb_n_type n
);
```

Parameters

x

n

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

signbit

```
long signbit (
    float x
);
```

Parameters

x

A value of type `float` or `double`, NaN, infinity, or zero.

Return Value
Returns a non-zero value only if the sign of the argument is negative.

Discussion
This function is implemented as an inline macro.

Availability
Available in Mac OS X version 10.0 and later.

Declared In
`fp.h`

sin

```
double_t sin (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

sinh

```
double_t sinh (
   double_t x
);
```

Parameters
x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

SInt64ToUInt64

```
UInt64 SInt64ToUInt64 (
   SInt64 value
);
```

Parameters
value

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

sqrt

```
double_t sqrt (
   double_t x
);
```

Parameters
x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

str2dec

```
void str2dec (
   const char *s,
   short *ix,
   decimal *d,
   short *vp
);
```

Parameters

s

ix

d

vp

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

tan

```
double_t tan (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

tanh

```
double_t tanh (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

trunc

```
_trunc_return_type trunc (
   double_t x
);
```

Parameters

x

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

U32SetU

```
UInt32 U32SetU (
   UInt64 value
);
```

Parameters

value

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64Add

```
UInt64 U64Add (
   UInt64 left,
   UInt64 right
);
```

Parameters

x

y

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64And

```
Boolean U64And (
   UInt64 left,
   UInt64 right
);
```

Parameters

left

right

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64BitwiseAnd

```
UInt64 U64BitwiseAnd (
   UInt64 left,
   UInt64 right
);
```

Parameters

left

right

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64BitwiseEor

```
UInt64 U64BitwiseEor (
   UInt64 left,
   UInt64 right
);
```

Parameters

left

right

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64BitwiseNot

```
UInt64 U64BitwiseNot (
   UInt64 value
);
```

Parameters

value

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64BitwiseOr

```
UInt64 U64BitwiseOr (
   UInt64 left,
   UInt64 right
);
```

Parameters

left

right

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64Compare

```
SInt32 U64Compare (
   UInt64 left,
   UInt64 right
);
```

Parameters

left

right

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64Div

Under evaluation

```
UInt64 U64Div (
   UInt64 dividend,
   UInt64 divisor
);
```

Parameters

dividend

divisor

Return Value

Availability

Declared In
Math64.h

U64Divide

```
UInt64 U64Divide (
   UInt64 dividend,
   UInt64 divisor,
   UInt64 *remainder
);
```

Parameters

dividend

divisor

remainder

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64Eor

```
Boolean U64Eor (
   UInt64 left,
   UInt64 right
);
```

Parameters

left

right

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64Max

```
UInt64 U64Max (
   void
);
```

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64Multiply

```
UInt64 U64Multiply (
   UInt64 left,
   UInt64 right
);
```

Parameters

xparam

yparam

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64Not

```
Boolean U64Not (
   UInt64 value
);
```

Parameters

value

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64Or

```
Boolean U64Or (
   UInt64 left,
   UInt64 right
);
```

Parameters

left

right

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64Set

```
UInt64 U64Set (
   SInt32 value
);
```

Parameters

value

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64SetU

```
UInt64 U64SetU (
   UInt32 value
);
```

Parameters

value

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64ShiftLeft

```
UInt64 U64ShiftLeft (
   UInt64 value,
   UInt32 shift
);
```

Parameters

value

shift

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64ShiftRight

```
UInt64 U64ShiftRight (
   UInt64 value,
   UInt32 shift
);
```

Parameters

value

shift

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

U64Subtract

```
UInt64 U64Subtract (
   UInt64 left,
   UInt64 right
);
```

Parameters

left

right

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

UInt64ToSInt64

```
SInt64 UInt64ToSInt64 (
   UInt64 value
);
```

Parameters

value

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
Math64.h

WideAdd

```
wide * WideAdd (
   wide *target,
   const wide *source
);
```

Parameters

target

source

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Related Sample Code
SoftVDigX

Declared In
FixMath.h

WideBitShift

```
wide * WideBitShift (
   wide *target,
   SInt32 shift
);
```

Parameters

src

shift

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
FixMath.h

WideCompare

```
short WideCompare (
   const wide *target,
   const wide *source
);
```

Parameters

target

source

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
FixMath.h

WideDivide

```
SInt32 WideDivide (
   const wide *dividend,
   SInt32 divisor,
   SInt32 *remainder
);
```

Parameters

dividend

divisor

remainder

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
FixMath.h

WideMultiply

```
wide * WideMultiply (
   SInt32 multiplicand,
   SInt32 multiplier,
   wide *target
);
```

Parameters

multiplicand

multiplier

target

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
FixMath.h

WideNegate

```
wide * WideNegate (
   wide *target
);
```

Parameters

target

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
FixMath.h

WideShift

```
wide * WideShift (
   wide *target,
   SInt32 shift
);
```

Parameters

target

shift

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
FixMath.h

WideSquareRoot

```
UInt32 WideSquareRoot (
   const wide *source
);
```

Parameters

source

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
FixMath.h

WideSubtract

```
wide * WideSubtract (
   wide *target,
   const wide *source
);
```

Parameters

target

source

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
FixMath.h

WideWideDivide

```
wide * WideWideDivide (
   wide *dividend,
   SInt32 divisor,
   SInt32 *remainder
);
```

Parameters

dividend

divisor

remainder

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
FixMath.h

X2Fix

Converts an `Extended` number to a `Fixed` number.

```
Fixed X2Fix (
   double x
);
```

Parameters

x

The `Extended` number to be converted to a `Fixed` number.

Return Value

The best `Fixed` approximation of the `Extended` number `x`. If `x` is greater than the maximum representable `Fixed` number, the `X2Fix` function returns $7FFFFFFF. If `x` is less than the negative number with the highest absolute value, `X2Fix` returns $80000000.

Availability

Available in Mac OS X version 10.0 and later.

Related Sample Code

LiveVideoMixer2

Declared In

`FixMath.h`

X2Frac

Converts an `Extended` number to a `Fract` number.

```
Fract X2Frac (
   double x
);
```

Parameters

x

The `Extended` number to be converted to a `Fract` number.

Return Value

The best `Fract` approximation of the `Extended` number `x`. If `x` is greater than the maximum representable `Fract` number, the `X2Frac` function returns $7FFFFFFF. If `x` is less than the negative number with the highest absolute value, `X2Frac` returns $80000000.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`FixMath.h`

x80tod

```
double x80tod (
   const extended80 *x80
);
```

Parameters

x80

Return Value

Availability
Available in Mac OS X version 10.0 and later.

Declared In
fp.h

Data Types

decform

```
struct decform {
    char style;
    char unused;
    short digits;
};
typedef struct decform decform;
```

Fields
style
unused
digits

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

Declared In
fp.h

decimal

```
struct decimal {
    char sgn
    char unused
    short exp
    struct {
        unsigned char length;
        unsigned char text[36];
        unsigned char pad;
    } sig;
};
typedef struct decimal decimal;
```

Fields

`sgn`

`unused`

`exp`

`length`

`text`

`pad`

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

Declared In
`fp.h`

double_t

```
typedef double double_t;
```

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

Declared In
`fp.h`

fenv_t

```
typedef SInt32 fenv_t;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.1.

Declared In
`fenv.h`

fexcept_t

```
typedef SInt32 fexcept_t;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.1.

Declared In
`fenv.h`

Fixed

Defines a data type for fixed-point decimal numbers.

```
typedef SInt32 Fixed;
```

Discussion
This data type uses a 16-bit signed integer and a 16-bit fraction to represent fixed-point decimal numbers in the interval:

$$[-32768, 32767 + ((2^{16} - 1) / 2^{16})]$$

For example, the number 1.5 would be represented as `0x00018000`, and the number -1.3 would be represented as `0xFFFEB334`. To convert numbers between `Fixed` and `float`, you can use the functions `FixedToFloat` (page 1300) and `FloatToFixed` (page 1302).

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

Declared In
`IOMacOSTypes.h`

Fract

Defines a high-precision data type for fixed-point decimal numbers.

```
typedef SInt32 Fract;
```

Discussion
This data type uses a 2-bit signed integer and a 30-bit fraction to represent fixed-point decimal numbers in the interval

$$[-2, 1 + ((2^{30} - 1) / 2^{30})]$$

with higher precision than the `Fixed` (page 1347) data type. For example, the number 1.5 would be represented as `0x60000000`, and the number -1.3 would be represented as `0xACCCCCCD`. To convert numbers between `Fract` and `float`, you can use the functions `FractToFloat` (page 1308) and `FloatToFract` (page 1303).

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

Declared In
IOMacOSTypes.h

float_t

```
typedef float float_t;
```

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

Declared In
fp.h

relop

```
typedef short relop;
```

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

Declared In
fp.h

_scalb_n_type

```
typedef int _scalb_n_type;
```

_trunc_return_type

```
typedef double_t _trunc_return_type;
```

Constants

DECSTROUTLEN

```
enum {
    DECSTROUTLEN = 80
};
```

Constants
DECSTROUTLEN

FE_INEXACT

Definitions of floating-point exception macros.

```
enum {
    FE_INEXACT                  = 0x02000000,
    FE_DIVBYZERO                = 0x04000000,
    FE_UNDERFLOW                = 0x08000000,
    FE_OVERFLOW                 = 0x10000000,
    FE_INVALID                  = 0x20000000,
    FE_ALL_EXCEPT               = 0x3E000000
};
```

Constants

FE_INEXACT
Available in Mac OS X v10.0 through Mac OS X v10.1.
Declared in fenv.h.

FE_DIVBYZERO
Available in Mac OS X v10.0 through Mac OS X v10.1.
Declared in fenv.h.

FE_UNDERFLOW
Available in Mac OS X v10.0 through Mac OS X v10.1.
Declared in fenv.h.

FE_OVERFLOW
Available in Mac OS X v10.0 through Mac OS X v10.1.
Declared in fenv.h.

`FE_INVALID`
Available in Mac OS X v10.0 through Mac OS X v10.1.
Declared in `fenv.h`.

`FE_ALL_EXCEPT`
Available in Mac OS X v10.1 through Mac OS X v10.1.
Declared in `fenv.h`.

FE_LDBLPREC

```
enum {
    FE_LDBLPREC = 0,
    FE_DBLPREC = 1,
    FE_FLTPREC = 2
};
```

Constants
`FE_LDBLPREC`
`FE_DBLPREC`
`FE_FLTPREC`

FE_TONEAREST

Definitions of rounding direction macros.

```
enum {
    FE_TONEAREST                = 0x00000000,
    FE_TOWARDZERO               = 0x00000001,
    FE_UPWARD                   = 0x00000002,
    FE_DOWNWARD                 = 0x00000003
};
```

Constants
`FE_TONEAREST`
Available in Mac OS X v10.0 through Mac OS X v10.1.
Declared in `fenv.h`.

`FE_TOWARDZERO`
Available in Mac OS X v10.0 through Mac OS X v10.1.
Declared in `fenv.h`.

`FE_UPWARD`
Available in Mac OS X v10.0 through Mac OS X v10.1.
Declared in `fenv.h`.

`FE_DOWNWARD`
Available in Mac OS X v10.0 through Mac OS X v10.1.
Declared in `fenv.h`.

fixed1

```
enum {
    fixed1 = 0x00010000,
    fract1 = 0x40000000,
    positiveInfinity = 0x7FFFFFFF,
    negativeInfinity = 0x80000000
};
```

Constants

`fixed1`

`fract1`

`positiveInfinity`

`negativeInfinity`

FP_SNAN

```
enum {
    FP_SNAN = 0,
    FP_QNAN = 1,
    FP_INFINITE = 2,
    FP_ZERO = 3,
    FP_NORMAL = 4,
    FP_SUBNORMAL = 5
};
```

Constants

`FP_SNAN`

Available in Mac OS X v10.0 through Mac OS X v10.1.

Declared in `fp.h`.

`FP_QNAN`

Available in Mac OS X v10.0 through Mac OS X v10.1.

Declared in `fp.h`.

`FP_INFINITE`

Available in Mac OS X v10.0 through Mac OS X v10.1.

Declared in `fp.h`.

`FP_ZERO`

Available in Mac OS X v10.0 through Mac OS X v10.1.

Declared in `fp.h`.

`FP_NORMAL`

Available in Mac OS X v10.0 through Mac OS X v10.1.

Declared in `fp.h`.

`FP_SUBNORMAL`

Available in Mac OS X v10.0 through Mac OS X v10.1.

Declared in `fp.h`.

Relational Operator

```
typedef short relop;
enum {
    GREATERTHAN = 0,
    LESSTHAN = 1,
    EQUALTO = 2,
    UNORDERED = 3
};
```

Constants

`GREATERTHAN`
Available in Mac OS X v10.0 and later.
Declared in `fp.h`.

`LESSTHAN`
Available in Mac OS X v10.0 and later.
Declared in `fp.h`.

`EQUALTO`
Available in Mac OS X v10.0 and later.
Declared in `fp.h`.

`UNORDERED`
Available in Mac OS X v10.0 and later.
Declared in `fp.h`.

SIGDIGLEN

```
enum {
    SIGDIGLEN = 36
};
```

Constants

`SIGDIGLEN`

Special Values

```
#define HUGE_VAL
#define INFINITY
```

Constants

`HUGE_VAL`
Available in Mac OS X v10.0 and later.
Declared in `fp.h`.

`INFINITY`
Available in Mac OS X v10.0 and later.
Declared in `fp.h`.

Memory Management Utilities Reference

Framework: CoreServices/CoreServices.h
Declared in OSUtils.h

Overview

Applications can use the Memory Management Utilities to

- ensure that their callback routines, interrupt tasks, and stand-alone code could access application global variables or QuickDraw global variables
- add elements to and remove them from an operating-system queue
- ensure that they function properly in both 24- and 32-bit modes
- ensure that data or instructions in the microprocessor's internal caches remain consistent with data or instructions in RAM

While Carbon supports most of the Memory Management Utilities, there are changes to functions that assume a 68K runtime environment.

- Functions that flush caches on 68K processors (such as `FlushInstructionCache`, `FlushDataCache`, and `FlushCodeCacheRange`) are no longer supported.
- Functions such as `SetA5` or `SetCurrentA5` do nothing when running in Mac OS X. However, these functions should work normally when running in Mac OS 8 or 9.
- The functions `GetMMUMode` and `SwapMMUMode` are not supported because all PowerPC applications use 32-bit addressing, even if they are not Carbon-compliant.
- The `SysEnvirons` function is no longer supported since the Gestalt Manager can provide the same information. You should call the functions `FindFolder` and `Gestalt` instead.

For a list of unsupported functions, see the Carbon Specification.

Functions by Task

Determining the Measurement System

`IsMetric` (page 1361)
Verifies whether the current script system is using the metric system or the English system of measurement.

Reading and Writing Location Data

`ReadLocation` (page 1362)
Obtains information about a geographic location or time zone.

`WriteLocation` (page 1365) Deprecated in Mac OS X v10.0
Changes the geographic location or time-zone information stored in extended parameter RAM. (Deprecated. There is no replacement because you cannot set this information in Mac OS X.)

Setting and Restoring the A5 Register

`SetA5` (page 1363) Deprecated in Mac OS X v10.4
Sets the A5 register to the address specified. (Deprecated. There is no replacement because Mac OS X doesn't use the A5 variable.)

`SetCurrentA5` (page 1364) Deprecated in Mac OS X v10.4
Sets the value in register A5 to the value of the low-memory global variable CurrentA5. (Deprecated. There is no replacement because Mac OS X doesn't use the A5 variable.)

Getting the User and Computer Name

`CSCopyMachineName` (page 1355)
Returns a reference to the CFString that represents the computer name.

`CSCopyUserName` (page 1356)
Returns a reference to the CFString that represents the user name.

Managing a Queue

`Enqueue` (page 1359)
Adds elements directly to an operating-system queue or a queue that you create.

`Dequeue` (page 1356)
Removes a queue element directly from an operating-system queue or from a queue that you have created.

`DTUninstall` (page 1358)
(Deprecated. You should restructure your application to use threads, such as those supplied by Multiprocessing Services.)

`DTInstall` (page 1358) Deprecated in Mac OS X v10.4
Adds the specified task record to the deferred-task queue. (Deprecated. You should restructure your application to use threads, such as those supplied by Multiprocessing Services.)

Working With Parameter RAM

`InitUtil` (page 1360) Deprecated in Mac OS X v10.3
Copies the contents of parameter RAM into low memory. (Deprecated. There is no replacement because Mac OS X doesn't require this initialization.)

`GetSysPPtr` (page 1360) Deprecated in Mac OS X v10.4
: Returns a pointer to the low-memory copy of parameter RAM. (Deprecated. There is no replacement; this function always returns `NULL` in Mac OS X.)

`WriteParam` (page 1366) Deprecated in Mac OS X v10.4
: Write the modified values in the system parameters data structure to parameter RAM. (Deprecated. There is no replacement, because this function does nothing in Mac OS X.)

Miscellaneous

`Delay` (page 1356)
: Delays execture for the specified amount of time.

`TickCount` (page 1365)
: Obtains the current number of ticks (a tick is approximately 1/60 of a second) since the system last started up.

`MakeDataExecutable` (page 1362)
: Notifies the system that the specified data is subject to execution.

Working With Universal Procedure Pointers

`NewDeferredTaskUPP` (page 1362)
: Creates a new universal procedure pointer (UPP) to a deferred-task callback.

`InvokeDeferredTaskUPP` (page 1361)
: Calls a deferred-task callback.

`DisposeDeferredTaskUPP` (page 1358)
: Disposes of a universal procedure pointer (UPP) to a deferred-task callback.

Functions

CSCopyMachineName

Returns a reference to the CFString that represents the computer name.

```
CFStringRef CSCopyMachineName (
   void
);
```

Return Value
A `CFStringRef`. See the Base Services documentation for a description of the `CFStringRef` data type.

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

Declared In
`OSUtils.h`

CSCopyUserName

Returns a reference to the CFString that represents the user name.

```
CFStringRef CSCopyUserName (
   Boolean useShortName
);
```

Parameters

useShortName

A Boolean value that specifies whether to return the short name or full name of the user.

Return Value

A `CFStringRef`. See the Base Services documentation for a description of the `CFStringRef` data type.

Discussion

The function `CSCopyUserName` returns a `CFStringRef` based on the read UID (RUID, as returned by `getuid`) of the calling process. This can result in unexpected behavior (that is, `CSCopyUserName` returning different results than `SCDynamicStoreCopyConsoleUser`) for processes that manipulate their UID.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

Delay

Delays execture for the specified amount of time.

```
void Delay (
   unsigned long numTicks,
   unsigned long *finalTicks
);
```

Parameters

numTicks

finalTicks

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

Dequeue

Removes a queue element directly from an operating-system queue or from a queue that you have created.

```
OSErr Dequeue (
   QElemPtr qElement,
   QHdrPtr qHeader
);
```

Parameters

qElement

A pointer to a queue element to remove from a queue.

qHeader

A pointer to a queue header.

Return Value

A result code. See "Memory Management Utilities Result Codes" (page 1376).

Discussion

The `Dequeue` function attempts to find the queue element specified by the `qElement` parameter in the queue specified by the `qHeader` parameter. If `Dequeue` finds the element, it removes the element from the queue, adjusts the other elements in the queue accordingly, and returns `noErr`. Otherwise, it returns `qErr`, indicating that it could not find the element in the queue. The `Dequeue` function does not deallocate the memory occupied by the queue element.

For a description of the `QElem` record, see `QElem` (page 1369); for a description of the `QHdr` record, see `QHdr` (page 1370).

The `Dequeue` function disables interrupts as it searches through the queue for the element to be removed. The time during which interrupts are disabled depends on the length of the queue and the position of the entry in the queue. The `Dequeue` function can be called at interrupt time. However, the `Dequeue` function is ordinarily used only by system software and, whenever possible, you should manipulate an operating-system queue indirectly, by calling special-purpose removal functions. You can use the Queue Utilities functions for directly manipulating queues that you create. Use the following functions instead of `Dequeue`:

- `SlotVRemove`

  Removes a slot-based VBL task. This function is available with the Vertical Retrace Manager.

- `VRemove`

  Removes a system-based VBL task. This function is available with the Vertical Retrace Manager.

- `WaitNextEvent`

  Removes an Event. This function is available with the Event manager.

- `SIntRemove`

  Removes a slot interrupt task. This function is available with the Slot Manager.

- `NMRemove`

  Removes a Notification request. This function is available with the Notification Manager.

- `SleepQRemove`

  Removes a Sleep task. This function is available with the Power Manager.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

DisposeDeferredTaskUPP

Disposes of a universal procedure pointer (UPP) to a deferred-task callback.

```
void DisposeDeferredTaskUPP (
   DeferredTaskUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion

See the callback `DeferredTaskProcPtr` (page 1367) for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

DTInstall

Adds the specified task record to the deferred-task queue. (Deprecated in Mac OS X v10.4. You should restructure your application to use threads, such as those supplied by Multiprocessing Services.)

```
OSErr DTInstall (
   DeferredTaskPtr dtTaskPtr
);
```

Parameters

dtTaskPtr

Return Value

A result code. See "Memory Management Utilities Result Codes" (page 1376).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`OSUtils.h`

DTUninstall

(Deprecated in Mac OS X v10.4. You should restructure your application to use threads, such as those supplied by Multiprocessing Services.)

```
OSErr DTUninstall (
   DeferredTaskPtr dtTaskPtr
);
```

Availability

Available in Mac OS X v10.4 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OSUtils.h`

Enqueue

Adds elements directly to an operating-system queue or a queue that you create.

```
void Enqueue (
   QElemPtr qElement,
   QHdrPtr qHeader
);
```

Parameters

qElement
A pointer to the queue element to add to a queue.

qHeader
A pointer to a queue header.

Discussion
The `Enqueue` function adds the queue element specified by the `qElement` parameter to the end of the queue specified by the `qHeader` parameter. The specified queue header is updated to reflect the new queue element.

For a description of the `QElem` record, see `QElem` (page 1369); for a description of the `QHdr` record, see `QHdr` (page 1370).

Because interrupt functions are likely to manipulate operating-system queues, interrupts are disabled for a short time while the specified queue is updated. You can call the `Enqueue` function at interrupt time. However, the `Enqueue` function is ordinarily used only by system software. Whenever possible, you should manipulate an operating-system queue indirectly, by calling special-purpose functions whenever possible, instead of the `Enqueue` function. You can use the Queue Utilities functions for directly manipulating queues that you create. Use the following functions instead of `Enqueue`:

- `SlotVInstall`

  Installs a slot-based VBL task. This function is available with the Vertical Retrace Manager.

- `VInstall`

  Installs a system-based VBL task. This function is available with the Vertical Retrace Manager.

- `AddDrive`

  Adds a disk drive. This function is available with the Device Manager.

- `PPostEvent` and `PostEvent`

  Installs an Event. This function is available with the Event manager.

- `DTInstall`

  Installs a deferred task. This function is available with the Memory Management Utilities.

- `SIntInstall`

  Installs a slot interrupt task. This function is available with the Slot Manager.

- `NMInstall`

  Installs a Notification request. This function is available with the Notification Manager.

- `SleepQInstall`

  Installs a Sleep task. This function is available with the Power Manager.

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

Declared In
`OSUtils.h`

GetSysPPtr

Returns a pointer to the low-memory copy of parameter RAM. (Deprecated in Mac OS X v10.4. There is no replacement; this function always returns `NULL` in Mac OS X.)

```
SysPPtr GetSysPPtr (
   void
);
```

Return Value
See the description of the `SysPPtr` 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
`OSUtils.h`

InitUtil

Copies the contents of parameter RAM into low memory. (Deprecated in Mac OS X v10.3. There is no replacement because Mac OS X doesn't require this initialization.)

```
OSErr InitUtil (
   void
);
```

Return Value
A result code. See "Memory Management Utilities Result Codes" (page 1376).

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
`OSUtils.h`

InvokeDeferredTaskUPP

Calls a deferred-task callback.

```
void InvokeDeferredTaskUPP (
   long dtParam,
   DeferredTaskUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeDeferredTaskUPP`, as the system calls your deferred-task callback for you. See the callback `DeferredTaskProcPtr` (page 1367) for more information.

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

Declared In
`OSUtils.h`

IsMetric

Verifies whether the current script system is using the metric system or the English system of measurement.

```
Boolean IsMetric (
   void
);
```

Return Value
`TRUE` if the metric system is being used; `FALSE` if the English system is being used.

Discussion
The `IsMetric` function examines the `metricSys` field of the numeric-format resource (resource type `'itl0'`) to determine if the current script is using the metric system. A value of 255 in the `metricSys` field indicates that the metric system (centimeters, kilometers, milligrams, degrees Celsius, and so on) is being used. A value of 0 in the `metricSys` field indicates that the English system of measurement (inches, miles, ounces, degrees Fahrenheit, and so on) is used.

If you want to use units of measurement different from that of the current script, you need to override the value of the `metricSys` field in the current numeric-format resource. You can do this by using your own version of the numeric-format resource instead of the current script system's default international resource.

The `IsMetric` function is the same as the `IUMetric` function, which was previously available with the International Utilities Package.

Special Considerations

The `IsMetric` function may move or purge blocks in the heap calling it may cause problems if you've dereferenced a handle. Do not call this function from within interrupt code, such as in a completion function or a VBL task.

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

Declared In
`OSUtils.h`

MakeDataExecutable

Notifies the system that the specified data is subject to execution.

```
void MakeDataExecutable (
   void *baseAddress,
   unsigned long length
);
```

Parameters

baseAddress

The starting address of the data to be flushed.

length

The length of the data pointed to by the `baseAddress` parameter.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`OSUtils.h`

NewDeferredTaskUPP

Creates a new universal procedure pointer (UPP) to a deferred-task callback.

```
DeferredTaskUPP NewDeferredTaskUPP (
   DeferredTaskProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your deferred-task callback.

Return Value

On return, a UPP to a deferred-task callback. See the description of the `DeferredTaskUPP` data type.

Discussion

See the callback `DeferredTaskProcPtr` (page 1367) for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

ReadLocation

Obtains information about a geographic location or time zone.

```
void ReadLocation (
   MachineLocation *loc
);
```

Parameters

loc

On return, the fields of the geographic location structure containing the geographic location and the time-zone information.The `ReadLocation` procedure reads the stored geographic location and time zone of the Macintosh computer from extended parameter RAM.

You can get values for the latitude, longitude, daylight savings time (DST), or Greenwich mean time (GMT). If the geographic location record has never been set, all fields contain 0.

Discussion

The latitude and longitude are stored as `Fract` values, giving accuracy to within one foot. For example, a `Fract` value of 1.0 equals 90 degrees –1.0 equals –90 degrees and –2.0 equals –180 degrees.

To convert these values to a degrees format, you need to convert the `Fract` values first to the `Fixed` data type, then to the `LongInt` data type. Use the Mathematical and Logical Utilities functions `Fract2Fix` and `Fix2Long` to accomplish this task.

The DST value is a signed byte value that specifies the offset for the `hour` field—whether to add one hour, subtract one hour, or make no change at all.

The GMT value is in seconds east of GMT. For example, San Francisco is at –28,800 seconds (8 hours * 3,600 seconds per hour) east of GMT. The `gmtDelta` field is a 3-byte value contained in a long word, so you must take care to get it properly.

The `ReadLocation` function was previously available with the Script Manager.

For more information on the geographic location record, see `MachineLocation` (page 1368).

For more information on the `Fract` data type and the conversion routines `Long2Fix`, `Fix2Fract`, `Fract2Fix`, and `Fix2Long`, see Mathematical and Logical Utilities.

Special Considerations

Do not call the `ReadLocation` function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

SetA5

Sets the A5 register to the address specified. (Deprecated in Mac OS X v10.4. There is no replacement because Mac OS X doesn't use the A5 variable.)

```
long SetA5 (
   long newA5
);
```

Parameters

newA5

The value to which the A5 register is to be changed.

Return Value
The value in the A5 register before SetA5 changes it to `newA5`.

Discussion
In interrupt code that accesses application global variables, use the `SetA5` function first to restore a value previously saved using `SetCurrentA5`, and then, at the end of the code, to restore the A5 register to the value it had before the first call to `SetA5`.

Carbon Porting Notes

68K-specific. Does nothing in PowerPC native 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
`OSUtils.h`

SetCurrentA5

Sets the value in register A5 to the value of the low-memory global variable CurrentA5. (Deprecated in Mac OS X v10.4. There is no replacement because Mac OS X doesn't use the A5 variable.)

```
long SetCurrentA5 (
   void
);
```

Return Value
The value in the A5 register before SetCurrentA5 changes it to the value of the low-memory global variable CurrentA5.

Discussion
The `CurrentA5` variable points to the boundary between the parameters and global variables of the current application.

You cannot reliably call `SetCurrentA5` in code that executes at interrupt time unless you first guarantee that your application is the current process (for example, by calling the Process Manager function `GetCurrentProcess`). In general, you should call `SetCurrentA5` at noninterrupt time and then pass the returned value to the interrupt code.

Carbon Porting Notes

68K-specific. Does nothing in PowerPC native 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
`OSUtils.h`

TickCount

Obtains the current number of ticks (a tick is approximately 1/60 of a second) since the system last started up.

```
UInt32 TickCount (
   void
);
```

Discussion
The `TickCount` function returns an unsigned 32-bit integer that indicates the current number of ticks since the system last started up. You can use this value to compare the number of ticks that have elapsed since a given event or other action occurred. For example, you could compare the current value returned by `TickCount` with the value of the `when` field of an event structure.

The tick count is incremented during the vertical retrace interrupt, but this interrupt can be disabled. Your application should not rely on the tick count to increment with absolute precision. Your application also should not assume that the tick count always increments by 1 an interrupt task might keep control for more than one tick. If your application keeps track of the previous tick count and then compares this value with the current tick count, your application should compare the two values by checking for a "greater than or equal" condition rather than "equal to previous tick count plus 1."

Do not rely on the tick count being exact; it is usually accurate to within one tick, but this level of accuracy is not guaranteed.

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

Related Sample Code
QTCarbonShell
Simple DrawSprocket

Declared In
`OSUtils.h`

WriteLocation

Changes the geographic location or time-zone information stored in extended parameter RAM. (Deprecated in Mac OS X v10.0. There is no replacement because you cannot set this information in Mac OS X.)

```
void WriteLocation (
   const MachineLocation *loc
);
```

Parameters
loc
The geographic location and time-zone information to write to the extended parameter RAM.

Discussion
The latitude and longitude are stored in the geographic location structure as `Fract` values, giving accuracy to within 1 foot. For example, a `Fract` value of 1.0 equals 90 degrees –1.0 equals –90 degrees and –2.0 equals –180 degrees.

Use the functions `Long2Fix` and `Fix2Fract` to convert longitude and latitude values to the `Fixed` data type and then to the `Fract` data type for storage.

Use the daylight savings time value signed byte value to specify the offset for the `hour` field—whether to add one hour, subtract one hour, or make no change at all.

The Greenwich mean time value is in seconds east of GMT. For example, San Francisco is at –28,800 seconds (8 hours * 3,600 seconds per hour) east of GMT.The `gmtDelta` field is a 3-byte value contained in a long word. When writing `gmtDelta`, mask off the top byte because it is reserved. Preserve the value of `dlsDelta`.

The `WriteLocation` function was previously available with the Script Manager.

For more information on the geographic location record, see `MachineLocation` (page 1368).

For more information on the `Fract` data type and the conversion routines `Long2Fix`, `Fix2Fract`, `Fract2Fix`, and `Fix2Long`, see Mathematical and Logical Utilities.

Special Considerations
Do not call the `WriteLocation` function at interrupt time.

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
`OSUtils.h`

WriteParam

Write the modified values in the system parameters data structure to parameter RAM. (Deprecated in Mac OS X v10.4. There is no replacement, because this function does nothing in Mac OS X.)

```
OSErr WriteParam (
   void
);
```

Return Value
A result code. See "Memory Management Utilities Result Codes" (page 1376).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OSUtils.h`

Callbacks

DeferredTaskProcPtr

Defines a pointer to a deferred-task callback.

```
typedef void (*DeferredTaskProcPtr) (
    long dtParam
);
```

If you name your function `MyDeferredTaskProc`, you would declare it like this:

```
void MyDeferredTaskProc (
    long dtParam
);
```

Parameters

dtParam

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

Data Types

DeferredTask

Contains information related to a deferred task.

```
struct DeferredTask {
    volatile QElemPtr qLink;
    short qType;
    volatile short dtFlags;
    DeferredTaskUPP dtAddr;
    long dtParam;
    long dtReserved;
};
typedef struct DeferredTask DeferredTask;
typedef DeferredTask * DeferredTaskPtr;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

DeferredTaskUPP

Defines a universal procedure pointer to a deferred-task callback.

```
typedef DeferredTaskProcPtr DeferredTaskUPP;
```

Discussion
For more information, see the description of the `DeferredTaskProcPtr` (page 1367) callback function.

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

Declared In
`OSUtils.h`

MachineLocation

Contains information about the geographical location of a computer.

```
struct MachineLocation {
    Fract latitude
    Fract longitude
    union {
    #if TARGET_RT_BIG_ENDIAN
        SInt8 dlsDelta;
    #endif
        long gmDelta;
        struct {
        #if TARGET_RT_LITTLE_ENDIAN
            SInt8 pad[3];
        #endif
            SInt8 Delta;
        } dls;
    } u;
};
typedef struct MachineLocation MachineLocation;
```

Fields

`latitude`
The location's latitude, in fractions of a great circle. For example, Copenhagen, Denmark is at 55.43 degrees north latitude. When writing the latitude to extended parameter RAM with the `WriteLocation` procedure, you must convert this value to a `Fract` data type. (For example, a `Fract` value of 1.0 equals 90 degrees –1.0 equals –90 degrees and –2.0 equals –180 degrees.) For more information on the `Fract` data type, see Mathematical and Logical Utilities.

`longitude`
The location's longitude, in fractions of a great circle. For example, Copenhagen, Denmark is at 12.34 degrees east longitude. When writing the longitude to extended parameter RAM with the `WriteLocation` procedure, you must convert this value to a `Fract` data type. (For example, a `Fract` value of 1.0 equals 90 degrees –1.0 equals –90 degrees and –2.0 equals –180 degrees.

`dlsDelta`
A value that represents the current state of daylight savings time.

`gmDelta`
`pad`
`delta`

A signed byte value representing the hour offset for daylight saving time. This field is a 1-byte value contained in a long word. It should be preserved when writing `gmtDelta`.

Discussion
The geographic location and time-zone information of a Macintosh computer are stored in extended parameter RAM. The `MachineLocation` data type defines the format for the geographic location record.

The `ReadLocation` and `WriteLocation` procedures use the geographic location record to read and store the geographic location and time zone information in extended parameter RAM. If the geographic location record has never been set, all fields contain 0.

In order for MachineLocation to be endian-safe, a new member has been added to the 'u' union in the structure. You are encouraged to use the new member instead of the old one.

If your code looked like this:

```
MachineLocation.u.dlsDelta = 1;
```

you should change it to this:

```
MachineLocation.u.dls.Delta = 1;
```

to be endian safe. The `gmtDelta` remains the same; the low 24-bits are used. Remember that order of assignment DOES matter.

This will overwrite results:

```
MachineLocation.u.dls.Delta = 0xAA;          // u = 0xAAGGGGGG; G=Garbage
MachineLocation.u.gmtDelta = 0xBBBBBB;       // u = 0x00BBBBBB;
```

when in fact reversing the assignment would have preserved the values:

```
MachineLocation.u.gmtDelta = 0xBBBBBB;       // u = 0x00BBBBB;
MachineLocation.u.dls.Delta = 0xAA;          // u = 0xAABBBBBB;
```

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

Declared In
`OSUtils.h`

QElem

Contains information about a queue element.

```
struct QElem {
    QElem * qLink;
    short qType;
    short qData[1];
};
typedef struct QElem QElem;
typedef QElem * QElemPtr;
```

Fields

`qLink`

The type of the queue element. For a description of the values which you can use in this field, see "Queue Types" (page 1374).

`qType`

A variable array of data. The type of data and the length depend upon the queue type, specified in the `qType` field.

`qData`

Discussion

A queue element is a single entry in a queue. Each operating-system queue created and maintained by the Macintosh Operating System consists of a queue header and a linked list of queue elements. The exact structure of an element in an operating-system queue depends on the type of the queue. The `QElem` data type defines the available queue elements.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

QHdr

Contains information about the event queue.

```
struct QHdr {
    volatile short qFlags;
    volatile QElemPtr qHead;
    volatile QElemPtr qTail;
};
typedef struct QHdr QHdr;
typedef QHdr * QHdrPtr;
```

Fields

`qFlags`

Queue flags.

`qHead`

First queue entry.

`qTail`

Last queue entry.

Discussion

The event queue consists of a header followed by the actual entries in the queue. The event queue has the same header as all standard Macintosh Operating System queues. The `Qhdr` structure defines the queue header.

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

Declared In
OSUtils.h

SysEnvRec

Contains information about the system environment.

```
struct SysEnvRec {
    short environsVersion;
    short machineType;
    short systemVersion;
    short processor;
    Boolean hasFPU;
    Boolean hasColorQD;
    short keyBoardType;
    short atDrvrVersNum;
    short sysVRefNum;
};
typedef struct SysEnvRec SysEnvRec;
```

Fields

`environsVersion`

The version number of the `SysEnvirons` function that was used to fill in the record.

When you call the `SysEnvirons` function, you specify a version number to ensure that you receive a system environment record that matches your expectations. If you request a more recent version of `SysEnvirons` than is available, `SysEnvirons` places its own version number in the `environsVersion` field and returns a function result `envVersTooBig`.

`machineType`

A code for the Macintosh model. See "Macintosh Model Codes" (page 1373). Use the `Gestalt` function to obtain information about machine types not listed among these constants.

`systemVersion`

The version number of the current System file, represented as two byte-long numbers with one or more implied decimal points. The value $0410, for example, represents system software version 4.1.

If you call `SysEnvirons` when a system earlier than 4.1 is running, the MPW glue places $0 in this field and returns a result code of `envNotPresent`.

`processor`

A code for the microprocessor. See "Microprocessor Codes" (page 1374).

`hasFPU`

A Boolean value that indicates whether hardware floating-point processing is available.

`hasColorQD`

A Boolean value that indicates whether Color QuickDraw is present. This field says nothing about the presence of a color monitor.

`keyBoardType`

A code for the keyboard type. See "Keyboard Constants" (page 1373). Use the `Gestalt` function to obtain information about keyboard types not listed among these constants.

If the Apple Desktop Bus is in use, this field returns the keyboard type of the keyboard on which the last keystroke was made.

`atDrvrVersNum`
: The version number of the AppleTalk driver (specifically, the .MPP driver) currently installed. If AppleTalk is not loaded, this field is 0.

`sysVRefNum`
: The working-directory reference number of the folder or volume that holds the open System file.

Discussion
The `SysEnvirons` function fills in a system environment record, which describes some aspects of the software and hardware environment.

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

Declared In
`OSUtils.h`

SysParmType

Contains settings used by the system at startup.

```
struct SysParmType {
    UInt8 valid;
    UInt8 aTalkA;
    UInt8 aTalkB;
    UInt8 config;
    short portA;
    short portB;
    long alarm;
    short font;
    short kbdPrint;
    short volClik;
    short misc;
};
typedef struct SysParmType SysParmType;
typedef SysParmType * SysPPtr;
```

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

Declared In
`OSUtils.h`

Constants

Addressing Errors

Specify a type of addressing error.

```
enum {
    false32b = 0,
    true32b = 1
};
```

Constants

`false32b`

Indicates a 24-bit addressing error.

Available in Mac OS X v10.0 and later.

Declared in `OSUtils.h`.

`true32b`

Indicates a 32-bit addressing error.

Available in Mac OS X v10.0 and later.

Declared in `OSUtils.h`.

Keyboard Constants

Specify keyboard types.

```
enum {
    envUnknownKbd = 0,
    envMacKbd = 1,
    envMacAndPad = 2,
    envMacPlusKbd = 3,
    envAExtendKbd = 4,
    envStandADBKbd = 5,
    envPrtblADBKbd = 6,
    envPrtblISOKbd = 7,
    envStdISOADBKbd = 8,
    envExtISOADBKbd = 9
};
```

Macintosh Model Codes

Specify models of Macintosh computers.

```
enum {
    envMac = -1,
    envXL = -2,
    envMachUnknown = 0,
    env512KE = 1,
    envMacPlus = 2,
    envSE = 3,
    envMacII = 4,
    envMacIIx = 5,
    envMacIIcx = 6,
    envSE30 = 7,
    envPortable = 8,
    envMacIIci = 9,
    envMacIIfx = 11
};
```

Microprocessor Codes

Specify types of microprocessors.

```
enum {
    envCPUUnknown = 0,
    env68000 = 1,
    env68010 = 2,
    env68020 = 3,
    env68030 = 4,
    env68040 = 5
};
```

Queue Types

Specifies queue types.

```
enum {
    dummyType = 0,
    vType = 1,
    ioQType = 2,
    drvQType = 3,
    evType = 4,
    fsQType = 5,
    sIQType = 6,
    dtQType = 7,
    nmType = 8
};
typedef SignedByte QTypes;
```

Constants

`dummyType`

Reserved.

Available in Mac OS X v10.0 and later.

Declared in `OSUtils.h`.

`vType`
Specifies a vertical retrace queue type. See the Vertical Retrace Manager for more information.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

`ioQType`
Specifies a file I/O or driver I/O queue type.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

`drvQType`
Specifies a drive queue type.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

`evType`
Specifies an event queue type. See the Event Manager for more information.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

`fsQType`
Specifies a volume-control-block queue type.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

`sIQType`
Specifies a slot interrupt queue type. See the Slot Manager for more information.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

`dtQType`
Specifies a deferred task queue type. See Memory Management Utilities for more information.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

`nmType`
Specifies a notification queue type. See the Notification Manager for more information.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

Discussion
The different queue types that are accessible to your application are defined by the `QTypes` data type. Each of these enumerated queue types determines a different type of queue element. These constants are used in the `qtype` field of the `QElem` (page 1369) structure.

Sorting Constants

Specify the result types for the function `RelString`.

```
enum {
    sortsBefore = -1,
    sortsEqual = 0,
    sortsAfter = 1
};
```

Constants

`sortsBefore`
Indicates the first string is less than the second string.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

`sortsEqual`
Indicates the first string is equivalent to the second string.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

`sortsAfter`
Indicates the first string is greater than the second string.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

Assorted Use Constants

Defines constants to indicate use of various things, such as to use MIDE or AppleTalk.

```
enum {
    useFree = 0,
    useATalk = 1,
    useAsync = 2,
    useExtClk = 3,
    useMIDI = 4
};
```

Version Number

Specifies the version of the current system environment.

```
enum {
    curSysEnvVers = 2
};
```

Result Codes

The most common result codes returned by Memory Management Utilities are listed in the table below. Memory Management Utilities may also return the following errors:

`noErr` (0)
`paramErr` (-50)

prWrErr (-87)
prInitErr (-88)
memROZErr (-99)
memFullErr (-108)
nilHandleErr (-109)
memWZErr (-111)
memPurErr (-112)
memBCErr (-115)
memLockedErr (-117)
notEnoughMemoryErr (-620)
notHeldErr (-621)
cannotMakeContiguousErr (-622)
notLockedErr (-623)
interruptsMaskedErr (-624)
cannotDeferErr (-625)

Result Code	Value	Description
qErr	-1	Queue element not found during deletion. Available in Mac OS X v10.0 and later.
vTypErr	-2	Invalid queue element. Available in Mac OS X v10.0 and later.
corErr	-3	Core routine number out of range Available in Mac OS X v10.0 and later.
unimpErr	-4	Unimplemented core routine. Available in Mac OS X v10.0 and later.
SlpTypeErr	-5	Invalid queue element. Available in Mac OS X v10.0 and later.
hwParamErr	-502	Processor does not support flushing a range. Available in Mac OS X v10.0 and later.

Memory Manager Reference

Framework: CoreServices/CoreServices.h

Declared in MacMemory.h

Overview

The Memory Manager was the memory management solution for versions of the Macintosh operating system prior to Mac OS X. It remains available for compatibility with legacy applications and for new applications that must work with legacy Carbon code that requires handles.

In previous versions of the Macintosh operating system, developers used the Memory Manager to set up an application's memory partition at launch time, to manage an application's heap, to minimize application memory fragmentation, and to implement a scheme to avoid low-memory conditions. All of these operations are now handled transparently by Mac OS X.

In the Mac OS X Memory Manager, many functions are deprecated, the callbacks are not functional, and the data types and constants are not used. In Mac OS X there is no need to set up or manage an application memory partition or to manage pointers. To allocate memory, in most cases you simply use the C functions `malloc` or `calloc`. Mac OS X ensures that every application has access to as much memory as it needs—up to 4 gigabytes of addressable space per 32-bit process.

Mac OS X does not support functions for accessing the system heap, as the system heap is unavailable to applications in Mac OS X. Starting with Mac OS X v10.3, Memory Manager is thread safe, and the `MemError` (page 1410) function now returns error codes on a per-thread basis.

For information on memory management issues when porting a legacy Macintosh application to Mac OS X, refer to the Carbon Porting Guide and to Technical Note 2130, Memory Allocation Recommendations on Mac OS X.

Functions by Task

Allocating and Releasing Nonrelocatable Blocks of Memory

`DisposePtr` (page 1391)
Releases memory occupied by a nonrelocatable block.

`NewPtr` (page 1416)
Allocates a nonrelocatable block of memory of a specified size.

`NewPtrClear` (page 1417)
Allocates a nonrelocatable block of memory of a specified size with all its bytes set to 0.

Allocating and Releasing Relocatable Blocks of Memory

`DisposeHandle` (page 1390)
Releases memory occupied by a relocatable block.

`NewEmptyHandle` (page 1413)
Initializes a new handle without allocating any memory for it to control.

`NewHandle` (page 1414)
Allocates a new relocatable memory block of a specified size in the current heap zone.

`NewHandleClear` (page 1415)
Allocates a relocatable block of memory of a specified size with all its bytes set to 0.

Allocating Temporary Memory

`TempNewHandle` (page 1430)
Allocates a new relocatable block of temporary memory.

Assessing Memory Conditions

`MemError` (page 1410)
Determines if an application's last direct call to a Memory Manager function executed successfully.

`LMGetMemErr` (page 1406)
Returns the result of the last Memory Manager function without clearing the value.

`LMSetMemErr` (page 1407)
Sets the value which will be returned by the `MemError` function.

`MaxBlock` (page 1409) Deprecated in Mac OS X v10.5
Returns a fixed value for block size that is compatible with most applications. (Deprecated. There is no replacement function; you can assume that any reasonable memory allocation will succeed.)

`StackSpace` (page 1427) Deprecated in Mac OS X v10.5
Returns the amount of space between the bottom of the stack and the top of the application heap. (Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

Changing the Sizes of Relocatable and Nonrelocatable Blocks

`GetHandleSize` (page 1394)
Returns the logical size of the relocatable block corresponding to a handle.

`GetPtrSize` (page 1395)
Returns the logical size of the nonrelocatable block corresponding to a pointer.

`SetHandleSize` (page 1425)
Changes the logical size of the relocatable block corresponding to the specified handle.

`SetPtrSize` (page 1426)
Changes the logical size of the nonrelocatable block corresponding to a pointer.

Managing Relocatable Blocks

`EmptyHandle` (page 1392)
Purges a relocatable block and sets the corresponding handle's master pointer to `NULL`.

`HLockHi` (page 1399)
Sets the lock bit on the block.

`ReallocateHandle` (page 1422)
Allocates a new relocatable block of a specified size and sets a handle's master pointer to point to the new block.

`RecoverHandle` (page 1423)
Returns a handle to a relocatable block pointed to by a specified `pointer`.

Manipulating Blocks of Memory

`BlockMove` (page 1385)
Copies a sequence of bytes from one location in memory to another.

`BlockMoveData` (page 1386)

`BlockMoveDataUncached` (page 1387)

`BlockMoveUncached` (page 1387)

`BlockZero` (page 1388)

`BlockZeroUncached` (page 1388)

`HandAndHand` (page 1396)
Concatenates two relocatable blocks.

`HandToHand` (page 1396)
Copies all of the data from one relocatable block to a new relocatable block.

`PtrAndHand` (page 1418)
Concatenates part or all of a memory block to the end of a relocatable block.

`PtrToHand` (page 1419)
Copies data referenced by a pointer to a new relocatable block.

`PtrToXHand` (page 1419)
Copies data referenced by a pointer to an existing relocatable block.

Setting the Properties of Relocatable Blocks

`HClrRBit` (page 1397)
Clears the resource flag of a relocatable block.

`HGetState` (page 1398)
Returns a signed byte representing the current properties of a relocatable block.

`HLock` (page 1398)
Prevents a relocatable block from moving within its heap zone.

`HSetRBit` (page 1402)
Sets the resource flag of a relocatable block.

`HSetState` (page 1402)
Restores the properties of a relocatable block.

`HUnlock` (page 1403)
Allows a relocatable block to move in its heap zone.

Miscellaneous

`IsHandleValid` (page 1405)
Checks that a handle is valid.

`IsHeapValid` (page 1405)
Always returns true in Mac OS X.

`IsPointerValid` (page 1406)
Checks that a pointer is valid.

Deprecated Functions

You should avoid using the functions listed in this section.

`FlushMemory` (page 1393)
Makes a portion of the address space clean. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

`LMGetApplZone` (page 1406)
(Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

`LMSetApplZone` (page 1407)
(Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

`FreeMem` (page 1393) Deprecated in Mac OS X v10.5
Returns the total amount of free space in the current heap zone. (Deprecated. There is no replacement function; you can assume that any reasonable memory allocation will succeed.)

`MaxMem` (page 1409) Deprecated in Mac OS X v10.5
Returns the size, in bytes, of the largest contiguous free block in the current heap zone. (Deprecated. There is no replacement function; you can assume that any reasonable memory allocation will succeed.)

`TempDisposeHandle` (page 1427) Deprecated in Mac OS X v10.5
Releases a relocatable block in the temporary heap. (Deprecated. Use `DisposeHandle` (page 1390) instead; Mac OS X does not have a separate temporary memory heap.)

`CheckAllHeaps` (page 1388) Deprecated in Mac OS X v10.4
Checks all known heaps for validity. (Deprecated. There is no replacement function; an application has access only to its own heap in Mac OS X.)

`CompactMem` (page 1389) Deprecated in Mac OS X v10.4
Compacts the heap by moving relocatable blocks as needed. (Deprecated. There is no replacement function; memory compaction is never needed and never performed in Mac OS X.)

DisposeGrowZoneUPP (page 1390) Deprecated in Mac OS X v10.4
(Deprecated. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never called.)

DisposePurgeUPP (page 1391) Deprecated in Mac OS X v10.4
(Deprecated. There is no replacement function; heaps are never purged in Mac OS X, so the purge function is never called.)

DisposeUserFnUPP (page 1392) Deprecated in Mac OS X v10.4
(Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

GetGrowZone (page 1394) Deprecated in Mac OS X v10.4
Returns the current heap zone's grow-zone function. (Deprecated. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never used.)

GZSaveHnd (page 1395) Deprecated in Mac OS X v10.4
Returns a relocatable block to be protected during grow-zone operations. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

HNoPurge (page 1400) Deprecated in Mac OS X v10.4
Marks a relocatable block as unpurgeable. (Deprecated. There is no replacement function; heaps are never purged in Mac OS X.)

HoldMemory (page 1400) Deprecated in Mac OS X v10.4
Makes a portion of the address space resident in physical memory and ineligible for paging. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

HPurge (page 1401) Deprecated in Mac OS X v10.4
Marks a relocatable block as purgeable. (Deprecated. There is no replacement function; heaps are never purged in Mac OS X.)

InvokeGrowZoneUPP (page 1403) Deprecated in Mac OS X v10.4
(Deprecated. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never called.)

InvokePurgeUPP (page 1404) Deprecated in Mac OS X v10.4
(Deprecated. There is no replacement function; heaps are never purged in Mac OS X, so the purge function is never called.)

InvokeUserFnUPP (page 1404) Deprecated in Mac OS X v10.4
(Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

LMGetSysZone (page 1407) Deprecated in Mac OS X v10.4
(Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

LMSetSysZone (page 1408) Deprecated in Mac OS X v10.4
(Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

MakeMemoryNonResident (page 1408) Deprecated in Mac OS X v10.4
Makes pages in the specified range immediately available for reuse. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

MakeMemoryResident (page 1408) Deprecated in Mac OS X v10.4
Makes a portion of the address space resident in physical memory. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

`MoreMasterPointers` (page 1411) Deprecated in Mac OS X v10.4

Allocates a specified number of master pointers in the current heap zone. (Deprecated. There is no replacement function; master pointers do not need to be pre-allocated in Mac OS X.)

`MoreMasters` (page 1411) Deprecated in Mac OS X v10.4

Allocates a block of master pointers in the current heap zone. (Deprecated. There is no replacement function; master pointers do not need to be pre-allocated in Mac OS X.)

`MoveHHi` (page 1412) Deprecated in Mac OS X v10.4

Moves a relocatable block as high in memory as possible. (Deprecated. There is no replacement function; there is no benefit to moving handles high in memory in Mac OS X.)

`NewGrowZoneUPP` (page 1414) Deprecated in Mac OS X v10.4

(Deprecated. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never called.)

`NewPurgeUPP` (page 1417) Deprecated in Mac OS X v10.4

(Deprecated. There is no replacement function; heaps are never purged in Mac OS X, so the purge function is never called.)

`NewUserFnUPP` (page 1418) Deprecated in Mac OS X v10.4

(Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

`PurgeMem` (page 1420) Deprecated in Mac OS X v10.4

Purges the current heap zone until the specified number of bytes are available. (Deprecated. There is no replacement; heaps are never purged in Mac OS X, so this function does nothing.)

`PurgeSpace` (page 1421) Deprecated in Mac OS X v10.4

Determines the total amount of free memory and the size of the largest allocatable block in the current heap zone if it were purged. (Deprecated. There is no replacement; heaps are never purged in Mac OS X.)

`PurgeSpaceContiguous` (page 1421) Deprecated in Mac OS X v10.4

(Deprecated. There is no replacement; heaps are never purged in Mac OS X.)

`PurgeSpaceTotal` (page 1422) Deprecated in Mac OS X v10.4

(Deprecated. There is no replacement; heaps are never purged in Mac OS X.)

`ReleaseMemoryData` (page 1423) Deprecated in Mac OS X v10.4

Releases the data of a portion of the address space. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

`ReserveMem` (page 1424) Deprecated in Mac OS X v10.4

Reserves space for a block of memory as close to the bottom of the current heap zone as possible. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

`SetGrowZone` (page 1425) Deprecated in Mac OS X v10.4

Specifies the current heap zone's grow-zone function. (Deprecated. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never called.)

`TempFreeMem` (page 1428) Deprecated in Mac OS X v10.4

Returns the maximum amount of free memory in the temporary heap. (Deprecated. There is no replacement function; Mac OS X does not have a separate temporary memory heap.)

`TempHLock` (page 1428) Deprecated in Mac OS X v10.4

Locks a relocatable block in the temporary heap. (Deprecated. Use `HLock` (page 1398) instead; Mac OS X does not have a separate temporary memory heap.)

`TempHUnlock` (page 1429) Deprecated in Mac OS X v10.4

Unlocks a relocatable block in the temporary heap. (Deprecated. Use `HUnlock` (page 1403) instead; Mac OS X does not have a separate temporary memory heap.)

`TempMaxMem` (page 1429) Deprecated in Mac OS X v10.4
: Returns the maximum amount of temporary memory available. (Deprecated. There is no replacement function; Mac OS X does not have a separate temporary memory heap.)

`TempTopMem` (page 1430) Deprecated in Mac OS X v10.4
: Returns the location of the top of the temporary heap. (Deprecated. There is no replacement function; Mac OS X does not have a separate temporary memory heap.)

`TopMem` (page 1431) Deprecated in Mac OS X v10.4
: Returns a pointer to the byte at the top of an application's partition. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

`UnholdMemory` (page 1431) Deprecated in Mac OS X v10.4
: Makes a currently held range of memory eligible for paging again. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

Functions

BlockMove

Copies a sequence of bytes from one location in memory to another.

```
static void BlockMove (
   const void *srcPtr,
   void *destPtr,
   Size byteCount
);
```

Parameters

srcPtr
: The address of the first byte to copy.

destPtr
: The destination address.

byteCount
: The number of bytes to copy. If the value of `byteCount` is 0, `BlockMove` does nothing.

Discussion

The `BlockMove` function copies the specified number of bytes from the address designated by srcPtr to that designated by `destPtr`. It updates no pointers.

The `BlockMove` function works correctly even if the source and destination blocks overlap.

You can safely call `BlockMove` at interrupt time. Even though it moves memory, `BlockMove` does not move relocatable blocks, but simply copies bytes.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

The `BlockMove` function currently flushes the processor caches whenever it moves more than 12 bytes. This behavior can adversely affect your application's performance. You might want to avoid calling `BlockMove` to move small amounts of data in memory if there is no possibility of moving stale data or instructions. For more information about stale data and instructions, see the discussion of the processor caches in the chapter "Memory Management Utilities" in *Inside Macintosh: Memory*.

Special Considerations

Beginning in Mac OS X v10.4, the `BlockMove` function is inlined to a direct call to the POSIX `memmove` function. For more information, see the header file `MacMemory.h`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
SoftVDigX

Declared In
`MacMemory.h`

BlockMoveData

```
static void BlockMoveData (
   const void *srcPtr,
   void *destPtr,
   Size byteCount
);
```

Parameters

srcPtr

destPtr

byteCount

Discussion
You should not make any assumptions about the state of the destination memory while `BlockMoveData` is executing. In the intermediate state, values may be present that are neither the original nor the final ones. For example, this function may use the 'dcbz' instruction. If the underlying memory is not cacheable, if the memory is write-through instead of copy-back, or if the cache block is flushed for some reason, the 'dcbz' instruction will write zeros to the destination. You can avoid the use of the 'dcbz' instruction by calling `BlockMoveDataUncached`, but even that function makes no other guarantees about the memory block's intermediate state.

Special Considerations

Beginning in Mac OS X v10.4, the `BlockMoveData` function is inlined to a direct call to the POSIX `memmove` function. For more information, see the header file `MacMemory.h`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTMetaData

Declared In
`MacMemory.h`

BlockMoveDataUncached

```
static void BlockMoveDataUncached (
   const void *srcPtr,
   void *destPtr,
   Size byteCount
);
```

Parameters

srcPtr

destPtr

byteCount

Discussion

You should not make any assumptions about the state of the destination memory while `BlockMoveDataUncached` is executing. In the intermediate state, values may be present that are neither the original nor the final ones.

Special Considerations

Beginning in Mac OS X v10.4, the `BlockMoveDataUncached` function is inlined to a direct call to the POSIX `memmove` function. For more information, see the header file `MacMemory.h`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

BlockMoveUncached

```
static void BlockMoveUncached (
   const void *srcPtr,
   void *destPtr,
   Size byteCount
);
```

Parameters

srcPtr

destPtr

byteCount

Special Considerations

Beginning in Mac OS X v10.4, the `BlockMoveUncached` function is inlined to a direct call to the POSIX `memmove` function. For more information, see the header file `MacMemory.h`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

BlockZero

```
static void BlockZero (
   void *destPtr,
   Size byteCount
);
```

Parameters

destPtr

byteCount

Special Considerations

Beginning in Mac OS X v10.4, the `BlockZero` function is inlined to a direct call to the POSIX `bzero` function. For more information, see the header file `MacMemory.h`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

Simple DrawSprocket

Declared In

`MacMemory.h`

BlockZeroUncached

```
static void BlockZeroUncached (
   void *destPtr,
   Size byteCount
);
```

Parameters

destPtr

byteCount

Special Considerations

Beginning in Mac OS X v10.4, the `BlockZeroUncached` function is inlined to a direct call to the POSIX `bzero` function. For more information, see the header file `MacMemory.h`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

CheckAllHeaps

Checks all known heaps for validity. (Deprecated in Mac OS X v10.4. There is no replacement function; an application has access only to its own heap in Mac OS X.)

```
Boolean CheckAllHeaps (
   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
MacMemory.h

CompactMem

Compacts the heap by moving relocatable blocks as needed. (Deprecated in Mac OS X v10.4. There is no replacement function; memory compaction is never needed and never performed in Mac OS X.)

```
Size CompactMem (
   Size cbNeeded
);
```

Parameters

cbNeeded
The size, in bytes, of the block for which CompactMem should attempt to make room.

Return Value
The size, in bytes, of the largest contiguous free block available after compacting the heap zone. CompactMem does not actually allocate that block.

Discussion
The Memory Manager automatically compacts the heap when a memory request fails. However, you can use the CompactMem function to compact the current heap zone manually.

CompactMem compacts the current heap zone not by purging blocks, but rather by moving unlocked, relocatable blocks down until they encounter nonrelocatable blocks or locked, relocatable blocks. CompactMem continues compacting until it either finds a contiguous block of at least cbNeeded free bytes or compacts the entire zone.

To compact the entire heap zone, call CompactMem(maxSize).

Call the function MemError (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Because CompactMem moves memory, you should not 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
MacMemory.h

DisposeGrowZoneUPP

(Deprecated in Mac OS X v10.4. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never called.)

```
void DisposeGrowZoneUPP (
   GrowZoneUPP userUPP
);
```

Parameters

userUPP

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

MacMemory.h

DisposeHandle

Releases memory occupied by a relocatable block.

```
void DisposeHandle (
   Handle h
);
```

Parameters

h

A handle to a relocatable block.

Discussion

The `DisposeHandle` function releases the memory occupied by the relocatable block whose handle is `h`. It also frees the handle's master pointer for other uses.

Do not use `DisposeHandle` to dispose of a handle obtained from the Resource Manager (for example, by a previous call to `GetResource`), use `ReleaseResource` instead. If, however, you have called `DetachResource` on a resource handle, you should dispose of the storage by calling `DisposeHandle`.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Special Considerations

After a call to `DisposeHandle`, all handles to the released block become invalid and should not be used again. Any subsequent calls to `DisposeHandle` using an invalid handle might crash your application.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ASCIIMoviePlayerSample

Gamma Filter for FxPlug and AE

QTCarbonShell

QTMetaData

WhackedTV

Declared In
MacMemory.h

DisposePtr

Releases memory occupied by a nonrelocatable block.

```
void DisposePtr (
   Ptr p
);
```

Parameters

p
A pointer to the nonrelocatable block you want to dispose of.

Discussion
When you no longer need a nonrelocatable block, call the `DisposePtr` function to free it for other uses.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

After a call to `DisposePtr`, all pointers to the released block become invalid and should not be used again. Any subsequent use of a pointer to the released block might cause a system error.

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

Related Sample Code
CarbonCocoa_PictureCursor
CarbonSketch
SoftVDigX

Declared In
MacMemory.h

DisposePurgeUPP

(Deprecated in Mac OS X v10.4. There is no replacement function; heaps are never purged in Mac OS X, so the purge function is never called.)

```
void DisposePurgeUPP (
   PurgeUPP userUPP
);
```

Parameters

userUPP

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
MacMemory.h

DisposeUserFnUPP

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void DisposeUserFnUPP (
   UserFnUPP userUPP
);
```

Parameters
userUPP

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
MacMemory.h

EmptyHandle

Purges a relocatable block and sets the corresponding handle's master pointer to NULL.

```
void EmptyHandle (
   Handle h
);
```

Parameters
h
A handle to a relocatable block.

Discussion
The EmptyHandle function purges the relocatable block whose handle is h and sets the handle's master pointer to NULL. The EmptyHandle function allows you to free memory taken by a relocatable block without freeing the relocatable block's master pointer for other uses. The block whose handle is h must be unlocked but need not be purgeable.

Note that if there are multiple handles to the relocatable block, then calling the EmptyHandle function empties them all, because all of the handles share a common master pointer. When you later use ReallocateHandle to reallocate space for the block, the master pointer is updated, and all of the handles reference the new block correctly.

To purge all of the blocks in a heap zone that are marked purgeable, use the PurgeMem (page 1420) function.

To free the memory taken up by a relocatable block and release the block's master pointer for other uses, use the DisposeHandle (page 1390) function.

Call the function MemError (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

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

Declared In
`MacMemory.h`

FlushMemory

Makes a portion of the address space clean. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
OSErr FlushMemory (
   void *address,
   unsigned long count
);
```

Return Value
This function always returns a value of `noErr`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

FreeMem

Returns the total amount of free space in the current heap zone. (Deprecated in Mac OS X v10.5. There is no replacement function; you can assume that any reasonable memory allocation will succeed.)

```
long FreeMem (
   void
);
```

Return Value
Returns a fixed value for heap size that is compatible with most applications.

Discussion
In Mac OS 8 and 9, this function returns the total amount of free space in the current heap zone. In Mac OS X, this function always returns a large fixed value because applications run in a large, protected memory space.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Special Considerations

Even though `FreeMem` does not move or purge memory, you should not call it at interrupt time because the heap might be in an inconsistent state.

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
MacMemory.h

GetGrowZone

Returns the current heap zone's grow-zone function. (Deprecated in Mac OS X v10.4. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never used.)

```
GrowZoneUPP GetGrowZone (
   void
);
```

Return Value
See the description of the GrowZoneUPP 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
MacMemory.h

GetHandleSize

Returns the logical size of the relocatable block corresponding to a handle.

```
Size GetHandleSize (
   Handle h
);
```

Parameters

h
A handle to a relocatable block.

Return Value
The logical size, in bytes, of the relocatable block whose handle is h. In case of error, the function return 0.

Discussion
Call the function MemError (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

You should not call GetHandleSize at interrupt time because the heap might be in an inconsistent state.

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

Related Sample Code
QTCarbonShell
SoftVDigX

Declared In
MacMemory.h

GetPtrSize

Returns the logical size of the nonrelocatable block corresponding to a pointer.

```
Size GetPtrSize (
   Ptr p
);
```

Parameters

p
A pointer to a nonrelocatable block.

Return Value
The logical size, in bytes, of the nonrelocatable block pointed to by p. In case of error, the function returns 0.

Discussion
Call the function MemError (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

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

Declared In
MacMemory.h

GZSaveHnd

Returns a relocatable block to be protected during grow-zone operations. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
Handle GZSaveHnd (
   void
);
```

Return Value
A handle to a block of memory that the Memory Manager reserves during grow-zone operations. Your grow-zone function must not move, purge, or delete this block. This function returns NULL if there is no such block.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
MacMemory.h

HandAndHand

Concatenates two relocatable blocks.

```
OSErr HandAndHand (
   Handle hand1,
   Handle hand2
);
```

Parameters

hand1

A handle to the first relocatable block, whose contents do not change but are concatenated to the end of the second relocatable block.

hand2

A handle to the second relocatable block, whose size the Memory Manager expands so that it can concatenate the information from `hand1` to the end of the contents of this block.

Return Value

A result code. See "Memory Manager Result Codes" (page 1443).

Discussion

The `HandAndHand` function concatenates the information from the relocatable block specified by `hand1` onto the end of the relocatable block specified by `hand2`. The `hand1` variable remains unchanged.

Because the `HandAndHand` function dereferences the handle `hand1`, you must call the `HLock` function to lock the block before calling `HandAndHand`. Afterward, you can call the `HUnlock` function to unlock it. Alternatively, you can save the block's original state by calling the `HGetState` function, lock the block by calling `HLock`, and then restore the original settings by calling `HSetState`.

Because `HandAndHand` moves memory, you should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacMemory.h`

HandToHand

Copies all of the data from one relocatable block to a new relocatable block.

```
OSErr HandToHand (
   Handle *theHndl
);
```

Parameters

theHndl

A handle to the relocatable block whose data `HandToHand` will copy. On return, `theHndl` contains a handle to a new relocatable block whose data duplicates the original.

Return Value

A result code. See "Memory Manager Result Codes" (page 1443).

Discussion

The `HandToHand` function attempts to copy the information in the relocatable block to which `theHndl` is a handle; if successful, `HandToHand` sets `theHndl` to a handle pointing to the new relocatable block.

If successful in creating a new relocatable block, the HandToHand function does not duplicate the properties of the original block. The new block is unlocked, unpurgeable, and not a resource. Call HLock, HPurge, or HSetRBit (or the combination of HGetState and HSetState) to adjust the properties of the new block.

To copy only part of a relocatable block into a new relocatable block, use the PtrToHand (page 1419) function. Before calling PtrToHand, lock and dereference the handle pointing to the relocatable block you want to copy.

Because HandToHand replaces its parameter with the new handle, you should retain the original parameter value somewhere else, otherwise you will not be able to access it. Here is an example:

```
Handle original, copy;
OSErr myErr;
...
copy = original;
        /*both handles access same block*/
myErr = HandToHand(copy);
        /*copy now points to copy of block*/
```

Because HandToHand allocates memory, you should not call it at interrupt time.

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

Declared In
MacMemory.h

HClrRBit

Clears the resource flag of a relocatable block.

```
void HClrRBit (
   Handle h
);
```

Parameters

h

A handle to a relocatable block. HClrRBit does nothing if the flag for the relocatable block pointed to by h is already cleared.

Discussion
The Resource Manager uses this function extensively, but you probably will not need to use it.

To disassociate the data in a resource handle from the resource file, you should use the Resource Manager function DetachResource instead of this function.

Call the function MemError (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

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

Declared In
MacMemory.h

HGetState

Returns a signed byte representing the current properties of a relocatable block.

```
SInt8 HGetState (
   Handle h
);
```

Parameters

h

A handle to a relocatable block.

Return Value

A signed byte (char) containing the flags of the master pointer for the given handle. In case of error, the value returned is meaningless.

Discussion

The `HGetState` function returns a signed byte (char) containing the flags of the master pointer for the given handle. You can save this byte, change the state of any of the flags using the functions described in this section, and then restore their original states by passing the byte to the `HSetState` function.

You can use bit-manipulation functions on the returned signed byte to determine the value of a given attribute. Currently the following bits are used:

If an error occurs during an attempt to get the state flags of the specified relocatable block, `HGetState` returns the low-order byte of the result code as its function result. For example, if the handle `h` points to a master pointer whose value is `NULL`, then the signed byte returned by `HGetState` will contain the value –109.

You may also call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

SoftVDigX

Declared In

`MacMemory.h`

HLock

Prevents a relocatable block from moving within its heap zone.

```
void HLock (
   Handle h
);
```

Parameters

h

A handle to a relocatable block.

Discussion

If you plan to dereference a handle and then allocate, move, or purge memory (or call a function that does so), then you should lock the handle before using the dereferenced handle.

If the block is already locked, `HLock` does nothing.

If you plan to lock a relocatable block for long periods of time, you can prevent fragmentation by ensuring that the block is as low as possible in the heap zone. To do this, see the description of the `ReserveMem` (page 1424) function.

If you plan to lock a relocatable block for short periods of time, you can prevent heap fragmentation by moving the block to the top of the heap zone before locking. For more information, see the description of the `MoveHHi` (page 1412) function.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

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

Related Sample Code
QTMetaData
SoftVDigX

Declared In
`MacMemory.h`

HLockHi

Sets the lock bit on the block.

```
void HLockHi (
   Handle h
);
```

Parameters

h
A handle to a relocatable block.

Discussion
The `HLockHi` function is an alternative to using the two functions `MoveHHi` (deprecated in Mac OS X) and `HLock`. Because the `MoveHHi` function does not move memory in Mac OS X, there is no benefit to using this function.

This function will not return a meaningful error code. If you call `HLockHi` on a locked handle, it will return `noErr` (not `memLockedErr`) because it is not an error to call `HLock` on a locked handle.

Do not call `HLockHi` on blocks in the system heap. Do not call `HLockHi` from a desk accessory.

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

Declared In
`MacMemory.h`

HNoPurge

Marks a relocatable block as unpurgeable. (Deprecated in Mac OS X v10.4. There is no replacement function; heaps are never purged in Mac OS X.)

```
void HNoPurge (
   Handle h
);
```

Parameters

h

A handle to a relocatable block.

Discussion

The `HNoPurge` function marks the relocatable block, to which `h` is a handle, as unpurgeable. If the block is already unpurgeable, `HNoPurge` does nothing.

The `HNoPurge` function does not reallocate memory for a handle if it has already been purged.

If you want to reallocate memory for a relocatable block that has already been purged, you can use the `ReallocateHandle` (page 1422) function.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

HoldMemory

Makes a portion of the address space resident in physical memory and ineligible for paging. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
OSErr HoldMemory (
   void *address,
   unsigned long count
);
```

Parameters

address

A pointer indicating the starting address of the range of memory to be held in RAM.

count

The size, in bytes, of the range of memory to be held in RAM.

Return Value

This function always returns a value of `noErr`.

Discussion
If the starting address you supply to the `HoldMemory` function is not on a page boundary, then `HoldMemory` rounds down to the nearest page boundary. Similarly, if the specified range does not end on a page boundary, `HoldMemory` rounds up the value you pass in the `count` parameter so that the entire range of memory is held.

Even though `HoldMemory` does not move or purge memory, you should not 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
`MacMemory.h`

HPurge

Marks a relocatable block as purgeable. (Deprecated in Mac OS X v10.4. There is no replacement function; heaps are never purged in Mac OS X.)

```
void HPurge (
   Handle h
);
```

Parameters

h

A handle to a relocatable block.

Discussion
The `HPurge` function marks the relocatable block, to which `h` is a handle, as purgeable. If the block is already purgeable, `HPurge` does nothing.

The Memory Manager might purge the block when it needs to purge the heap zone containing the block to satisfy a memory request. A direct call to the `MaxMem` function would also purge blocks marked as purgeable.

Once you mark a relocatable block as purgeable, you should make sure that handles to the block are not empty before you access the block. If they are empty, you must reallocate space for the block and recopy the block's data from another source, such as a resource file, before using the information in the block.

If the block to which `h` is a handle is locked, `HPurge` does not unlock the block but does mark it as purgeable. If you later call `HUnlock` on `h`, the block is subject to purging.

If the Memory Manager has purged a block, you can reallocate space for it by using the `ReallocateHandle` (page 1422) function.

You can immediately free the space taken by a handle without disposing of it by calling the function `EmptyHandle` (page 1392). This function does not require that the block be purgeable.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
MacMemory.h

HSetRBit

Sets the resource flag of a relocatable block.

```
void HSetRBit (
   Handle h
);
```

Parameters

h

A handle to a relocatable block. HSetRBit does nothing if the flag for the relocatable block pointed to by h is already set.

Discussion
The Resource Manager uses this function extensively, but you probably will not need to use it.

When the resource flag is set, the Resource Manager identifies the associated relocatable block as belonging to a resource. This can cause problems if that block wasn't actually read from a resource.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

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

Declared In
MacMemory.h

HSetState

Restores the properties of a relocatable block.

```
void HSetState (
   Handle h,
   SInt8 flags
);
```

Parameters

h

A handle to a relocatable block.

flags

A signed byte (char) specifying the properties to which you want to set the relocatable block.

Discussion
You can use HSetState to restore properties of a block after a call to HGetState. See the description of the `HGetState` function for a list of the currently used bits in the flags byte. Because additional bits of the `flags` byte could become significant in future versions of system software, use `HSetState` only with a byte returned

by HGetState. If you need to set two or three properties of a relocatable block at once, it is better to use the functions that set individual properties than to manipulate the bits returned by HGetState and then call HSetState.

Call the function MemError (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

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

Related Sample Code
SoftVDigX

Declared In
MacMemory.h

HUnlock

Allows a relocatable block to move in its heap zone.

```
void HUnlock (
   Handle h
);
```

Parameters

h
A handle to a relocatable block.

Discussion
The HUnlock function unlocks the relocatable block to which h is a handle, allowing the block to move within its heap zone. If the block is already unlocked, HUnlock does nothing.

Call the function MemError (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

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

Related Sample Code
QTMetaData
SoftVDigX

Declared In
MacMemory.h

InvokeGrowZoneUPP

(Deprecated in Mac OS X v10.4. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never called.)

```
long InvokeGrowZoneUPP (
   Size cbNeeded,
   GrowZoneUPP userUPP
);
```

Parameters

cbNeeded

userUPP

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

InvokePurgeUPP

(Deprecated in Mac OS X v10.4. There is no replacement function; heaps are never purged in Mac OS X, so the purge function is never called.)

```
void InvokePurgeUPP (
   Handle blockToPurge,
   PurgeUPP userUPP
);
```

Parameters

blockToPurge

userUPP

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

InvokeUserFnUPP

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void InvokeUserFnUPP (
   void *parameter,
   UserFnUPP userUPP
);
```

Parameters

parameter

userUPP

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
MacMemory.h

IsHandleValid

Checks that a handle is valid.

```
Boolean IsHandleValid (
   Handle h
);
```

Parameters

h
The handle to check.

Return Value
Returns `true` if the specified handle is valid. If the handle is `NULL` or if the handle refers to memory which was not properly allocated, `IsHandleValid` returns `false`. In Mac OS 8 and 9, `IsHandleValid` also returns `false` if the given handle is empty. In Mac OS X, however, zero-length blocks are considered valid and `IsHandleValid` returns `true` for an empty handle.

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

Declared In
MacMemory.h

IsHeapValid

Always returns true in Mac OS X.

```
Boolean IsHeapValid (
   void
);
```

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

Declared In
MacMemory.h

IsPointerValid

Checks that a pointer is valid.

```
Boolean IsPointerValid (
   Ptr p
);
```

Parameters

p

The pointer to check.

Return Value

Returns `true` if the specified pointer is valid. If the pointer is `NULL` or if the pointer points to memory which was not properly allocated, `IsPointerValid` returns `false`. In Mac OS 8 and 9, `IsPointerValid` also returns `false` if the given pointer points to a zero-length block in memory. In Mac OS X, however, zero-length blocks are considered valid and `IsPointerValid` returns `true`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacMemory.h`

LMGetApplZone

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
THz LMGetApplZone (
   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

`MacMemory.h`

LMGetMemErr

Returns the result of the last Memory Manager function without clearing the value.

```
SInt16 LMGetMemErr (
   void
);
```

Return Value

A result code. See "Memory Manager Result Codes" (page 1443).

Availability

Available in Mac OS X v10.0 and later.

Declared In
MacMemory.h

LMGetSysZone

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
THz LMGetSysZone (
   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
MacMemory.h

LMSetApplZone

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetApplZone (
   THz 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
MacMemory.h

LMSetMemErr

Sets the value which will be returned by the `MemError` function.

```
void LMSetMemErr (
   SInt16 value
);
```

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

Declared In
MacMemory.h

LMSetSysZone

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetSysZone (
   THz 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
`MacMemory.h`

MakeMemoryNonResident

Makes pages in the specified range immediately available for reuse. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
OSErr MakeMemoryNonResident (
   void *address,
   unsigned long count
);
```

Return Value
This function always returns a value of `noErr`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

MakeMemoryResident

Makes a portion of the address space resident in physical memory. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
OSErr MakeMemoryResident (
   void *address,
   unsigned long count
);
```

Return Value
A result code. See "Memory Manager Result Codes" (page 1443).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
MacMemory.h

MaxBlock

Returns a fixed value for block size that is compatible with most applications. (Deprecated in Mac OS X v10.5. There is no replacement function; you can assume that any reasonable memory allocation will succeed.)

```
long MaxBlock (
   void
);
```

Return Value
The maximum contiguous space, in bytes, that you could obtain after compacting the current heap zone. `MaxBlock` does not actually do the compaction.

Discussion
In Mac OS X, this function always returns a large value because virtual memory is always available to fulfill any request for memory.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

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
MacMemory.h

MaxMem

Returns the size, in bytes, of the largest contiguous free block in the current heap zone. (Deprecated in Mac OS X v10.5. There is no replacement function; you can assume that any reasonable memory allocation will succeed.)

```
Size MaxMem (
   Size *grow
);
```

Parameters

grow

On return, the maximum number of bytes by which the current heap zone can grow. After a call to `MaxApplZone`, `MaxMem` always sets this parameter to 0.

Return Value
The size, in bytes, of the largest contiguous free block in the zone after the compacting and purging.

Discussion
In Mac OS 8 and 9, the `MaxMem` function compacts the current heap zone and purges all relocatable, unlocked, and purgeable blocks from the zone. If the current zone is the original application zone, the `grow` parameter is set to the maximum number of bytes by which the zone can grow. For any other heap zone, `grow` is set to 0. `MaxMem` does not actually expand the zone or call the zone's grow-zone function.

In Mac OS X, the `MaxMem` function returns a large fixed value because applications run in a large, protected memory space.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

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
`MacMemory.h`

MemError

Determines if an application's last direct call to a Memory Manager function executed successfully.

```
OSErr MemError (
   void
);
```

Return Value
A result code. See "Memory Manager Result Codes" (page 1443).

Discussion
For each thread, `MemError` yields the result code produced by the last Memory Manager function your application called directly.

`MemError` is useful during application debugging. You might also use `MemError` as one part of a memory-management scheme to identify instances in which the Memory Manager rejects overly large memory requests by returning the error code `memFullErr`.

To view the result codes that `MemError` can produce, see "Memory Manager Result Codes" (page 1443).

Do not rely on `MemError` as the only component of a memory-management scheme. For example, suppose you call `NewHandle` or `NewPtr` and receive the result code `noErr`, indicating that the Memory Manager was able to allocate sufficient memory. In this case, you have no guarantee that the allocation did not deplete your application's memory reserves to levels so low that simple operations might cause your application to crash. Instead of relying on `MemError`, check before making a memory request that there is enough memory both to fulfill the request and to support essential operations.

Version Notes
Starting with Mac OS X v10.3, the `MemError` function provides error codes on a per-thread basis.

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

Related Sample Code
QTCarbonShell
SoftVDigX

Declared In
`MacMemory.h`

MoreMasterPointers

Allocates a specified number of master pointers in the current heap zone. (Deprecated in Mac OS X v10.4. There is no replacement function; master pointers do not need to be pre-allocated in Mac OS X.)

```
void MoreMasterPointers (
   UInt32 inCount
);
```

Parameters

inCount

The number of master pointers you want to allocate in a single nonrelocatable block.

Carbon Porting Notes

Carbon applications should use this function instead of `MoreMasters` to allocate a nonrelocatable block of master pointers in the current heap zone.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

MoreMasters

Allocates a block of master pointers in the current heap zone. (Deprecated in Mac OS X v10.4. There is no replacement function; master pointers do not need to be pre-allocated in Mac OS X.)

```
void MoreMasters (
   void
);
```

Discussion
In the application heap, a block of master pointers consists of 64 master pointers, and in the system heap, a block consists of 32 master pointers. (These values are likely to increase in future versions of system software.) When you initialize additional heap zones, you can specify the number of master pointers you want to have in a block of master pointers.

The Memory Manager automatically calls the `MoreMasters` function once for every new heap zone, including the application heap zone.

Call `MoreMasters` several times at the beginning of your program to prevent the Memory Manager from running out of master pointers in the middle of application execution. If it does run out, it allocates more, possibly causing heap fragmentation.

You should call `MoreMasters` at the beginning of your program enough times to ensure that the Memory Manager never needs to call it for you. For example, if your application never allocates more than 300 relocatable blocks in its heap zone, then five calls to the `MoreMasters` should be enough. It's better to call `MoreMasters` too many times than too few. For instance, if your application usually allocates about 100 relocatable blocks but might allocate 1000 in a particularly busy session, call `MoreMasters` enough times to accommodate the largest amount.

If you initialize a new zone, you can specify the number of master pointers that a master pointer block should contain.

Call the `MemError` (page 1410) function to get the result code. See "Memory Manager Result Codes" (page 1443).

Because `MoreMasters` allocates memory, you should not call it at interrupt time.

The calls to `MoreMasters` at the beginning of your application should be in the main code segment of your application or in a segment that the main segment never unloads.

Carbon Porting Notes

You should instead use `MoreMasterPointers` (page 1411).

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

`MacMemory.h`

MoveHHi

Moves a relocatable block as high in memory as possible. (Deprecated in Mac OS X v10.4. There is no replacement function; there is no benefit to moving handles high in memory in Mac OS X.)

```
void MoveHHi (
   Handle h
);
```

Parameters

h

A handle to a relocatable block.

Discussion

This function moves a relocatable block as high in memory as possible to help prevent heap fragmentation. The `MoveHHi` function attempts to move the relocatable block referenced by the handle `h` upward until it reaches a nonrelocatable block, a locked relocatable block, or the top of the heap.

If you plan to lock a relocatable block for a short period of time, use the `MoveHHi` function, which moves the block to the top of the heap and thus helps prevent heap fragmentation.

If you call `MoveHHi` to move a handle to a resource that has its `resChanged` bit set, the Resource Manager updates the resource by using the `WriteResource` function to write the contents of the block to disk. If you want to avoid this behavior, call the Resource Manager function `SetResPurge(FALSE)` before you call `MoveHHi`, and then call `SetResPurge(TRUE)` to restore the default setting.

By using the `MoveHHi` function on relocatable blocks you plan to allocate for short periods of time, you help prevent islands of immovable memory from accumulating in (and thus fragmenting) the heap.

Do not use the `MoveHHi` function to move blocks you plan to lock for long periods of time. The `MoveHHi` function moves such blocks to the top of the heap, perhaps preventing other blocks already at the top of the heap from moving down once they are unlocked. Instead, use the `ReserveMem` function before allocating such blocks, thus keeping them in the bottom partition of the heap, where they do not prevent relocatable blocks from moving.

If you frequently lock a block for short periods of time and find that calling `MoveHHi` each time slows down your application, you might consider leaving the block always locked and calling the `ReserveMem` function before allocating it.

Once you move a block to the top of the heap, be sure to lock it if you do not want the Memory Manager to move it back to the middle partition as soon as it can. (The `MoveHHi` function cannot move locked blocks; be sure to lock blocks after, not before, calling `MoveHHi`.)

Using the `MoveHHi` function without taking other precautionary measures to prevent heap fragmentation is useless, because even one small nonrelocatable or locked relocatable block in the middle of the heap might prevent `MoveHHi` from moving blocks to the top of the heap.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

NewEmptyHandle

Initializes a new handle without allocating any memory for it to control.

```
Handle NewEmptyHandle (
   void
);
```

Return Value
A handle with its master pointer set to `NULL`.

Discussion
The Resource Manager uses this function extensively, but you probably will not need to use it.

When you want to allocate memory for the empty handle, use the `ReallocateHandle` (page 1422) function.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

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

Declared In
`MacMemory.h`

NewGrowZoneUPP

(Deprecated in Mac OS X v10.4. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never called.)

```
GrowZoneUPP NewGrowZoneUPP (
   GrowZoneProcPtr userRoutine
);
```

Parameters
userRoutine

Return Value
See the description of the `GrowZoneUPP` 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
`MacMemory.h`

NewHandle

Allocates a new relocatable memory block of a specified size in the current heap zone.

```
Handle NewHandle (
   Size byteCount
);
```

Parameters
byteCount
The requested size, in bytes, of the relocatable block. Maximum size is 2 GB, the maximum size for variables of type `Size`.

Return Value
A handle to the new block. If `NewHandle` cannot allocate a block of the requested size, it returns `NULL`.

Discussion
The `NewHandle` function pursues all available avenues to create a block of the requested size, including compacting the heap zone, increasing its size, and purging blocks from it. If all of these techniques fail and the heap zone has a grow-zone function installed, `NewHandle calls the function`. Then `NewHandle`

tries again to free the necessary amount of memory, once more compacting and purging the heap zone if necessary. If `NewHandle` still cannot allocate memory, `NewHandle` calls the grow-zone function again, unless that function had returned 0, in which case `NewHandle` gives up and returns `NULL`.

If the `NewHandle` function succeeds in creating the requested block, this new block is unlocked and unpurgeable.

If you allocate a relocatable block that you plan to lock for long periods of time, you can prevent heap fragmentation by allocating the block as low as possible in the heap zone. To do this, see the description of the function `ReserveMem` (page 1424).

If you plan to lock a relocatable block for short periods of time, you might want to move it to the top of the heap zone to prevent heap fragmentation. For more information, see the description of the function `MoveHHi` (page 1412).

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Because `NewHandle` allocates memory, you should not call it at interrupt time.

Do not try to manufacture your own handles without this function by simply assigning the address of a variable of type `Ptr` to a variable of type `Handle`. The resulting "fake handle" would not reference a relocatable block and could cause a system crash.

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

Related Sample Code
Gamma Filter for FxPlug and AE
QTCarbonShell
QTMetaData
WhackedTV

Declared In
`MacMemory.h`

NewHandleClear

Allocates a relocatable block of memory of a specified size with all its bytes set to 0.

```
Handle NewHandleClear (
   Size byteCount
);
```

Parameters

byteCount
The requested size (in bytes) of the relocatable block. The `NewHandleClear` function sets each of these bytes to 0.

Return Value
A handle to the new block. If `NewHandleClear` cannot allocate a block of the requested size, it returns `NULL`.

Discussion
The `NewHandleClear` function works like the `NewHandle` function, but sets all bytes in the new block to 0 instead of leaving the contents of the block undefined.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Because `NewHandleClear` allocates memory, you should not call it at interrupt time.

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

Related Sample Code
LiveVideoMixer2
SoftVDigX

Declared In
`MacMemory.h`

NewPtr

Allocates a nonrelocatable block of memory of a specified size.

```
Ptr NewPtr (
   Size byteCount
);
```

Parameters

byteCount

The requested size (in bytes) of the nonrelocatable block. In Mac OS X, if you pass a value of zero, this function returns NULL, and `MemError` is set to `paramErr`. In Mac OS 9 and earlier, if you pass a value of zero, this function returns a valid zero length pointer.

Return Value
A pointer to the new block. If `NewPtr` fails to allocate a block of the requested size, it returns `NULL`.

Discussion
The `NewPtr` function attempts to reserve space as low in the heap zone as possible for the new block. If it is able to reserve the requested amount of space, `NewPtr` allocates the nonrelocatable block in the gap `ReserveMem` creates. Otherwise, `NewPtr` returns `NULL` and generates a `memFullErr` error.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Because `NewPtr` allocates memory, you should not call it at interrupt time.

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

Declared In
`MacMemory.h`

NewPtrClear

Allocates a nonrelocatable block of memory of a specified size with all its bytes set to 0.

```
Ptr NewPtrClear (
   Size byteCount
);
```

Parameters

byteCount

The requested size (in bytes) of the nonrelocatable block.

Return Value

A pointer to the new block. If NewPtrClear fails to allocate a block of the requested size, it returns `NULL`.

Discussion

The `NewPtrClear` function works much as the `NewPtr` function does, but sets all bytes in the new block to 0 instead of leaving the contents of the block undefined.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Because `NewPtrClear` allocates memory, you should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonCocoa_PictureCursor

CarbonSketch

SoftVDigX

Declared In

`MacMemory.h`

NewPurgeUPP

(Deprecated in Mac OS X v10.4. There is no replacement function; heaps are never purged in Mac OS X, so the purge function is never called.)

```
PurgeUPP NewPurgeUPP (
   PurgeProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value

See the description of the `PurgeUPP` 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
MacMemory.h

NewUserFnUPP

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
UserFnUPP NewUserFnUPP (
   UserFnProcPtr userRoutine
);
```

Parameters
userRoutine

Return Value
See the description of the UserFnUPP 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
MacMemory.h

PtrAndHand

Concatenates part or all of a memory block to the end of a relocatable block.

```
OSErr PtrAndHand (
   const void *ptr1,
   Handle hand2,
   long size
);
```

Parameters
ptr1
A pointer to the beginning of the data that the Memory Manager is to concatenate onto the end of the relocatable block.
hand2
A handle to the relocatable block, whose size the Memory Manager expands so that it can concatenate the information from ptr1 onto the end of this block.
size
The number of bytes of the block referenced by ptr1 to copy.

Return Value
A result code. See "Memory Manager Result Codes" (page 1443).

Discussion
The PtrAndHand function takes the number of bytes specified by the size parameter, beginning at the location specified by ptr1, and concatenates them onto the end of the relocatable block to which hand2 is a handle. The contents of the source block remain unchanged.

Because `PtrAndHand` allocates memory, you should not call it at interrupt time.

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

Related Sample Code
QTCarbonShell

Declared In
`MacMemory.h`

PtrToHand

Copies data referenced by a pointer to a new relocatable block.

```
OSErr PtrToHand (
   const void *srcPtr,
   Handle *dstHndl,
   long size
);
```

Parameters

srcPtr
The address of the first byte to copy.

dstHndl
A handle for which you have not yet allocated any memory. The `PtrToHand` function allocates memory for the handle and copies the specified number of bytes beginning at `srcPtr` into it. The dstHndl parameter is an output parameter that will hold the result. Its value on entry is ignored. If no error occurs, on exit it points to an unlocked, non-purgeable Handle of the requested size.

size
The number of bytes to copy.

Return Value
A result code. See "Memory Manager Result Codes" (page 1443).

Discussion
If you dereference and lock a handle, the `PtrToHand` function can copy its data to a new handle. However, for copying data from one handle to another, the `HandToHand` (page 1396) function is more efficient.

Because `PtrToHand` allocates memory, you should not call it at interrupt time.

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

Declared In
`MacMemory.h`

PtrToXHand

Copies data referenced by a pointer to an existing relocatable block.

```
OSErr PtrToXHand (
   const void *srcPtr,
   Handle dstHndl,
   long size
);
```

Parameters

srcPtr

The address of the first byte to copy.

dstHndl

A handle to an existing relocatable block.

size

The number of bytes to copy.

Return Value

A result code. See "Memory Manager Result Codes" (page 1443).

Discussion

The `PtrToXHand` function copies the specified number of bytes from the location specified by `srcPtr` to the handle specified by `dstHndl`.

Because `PtrToXHand` affects memory, you should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacMemory.h`

PurgeMem

Purges the current heap zone until the specified number of bytes are available. (Deprecated in Mac OS X v10.4. There is no replacement; heaps are never purged in Mac OS X, so this function does nothing.)

```
void PurgeMem (
   Size cbNeeded
);
```

Parameters

cbNeeded

The size, in bytes, of the block for which `PurgeMem` should attempt to make room.

Discussion

The Memory Manager purges the heap automatically when a memory request fails. However, you can use `PurgeMem` to purge the current heap zone manually.

The `PurgeMem` function sequentially purges blocks from the current heap zone until it either allocates a contiguous block of the specified size or purges the entire zone. If `PurgeMem` purges the entire zone without creating a contiguous block of the specified size, `PurgeMem` generates the result code `memFullErr`.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

The `PurgeMem` function purges only relocatable, unlocked, purgeable blocks. The function does not actually attempt to allocate the memory.

To purge the entire heap zone, call `PurgeMem(maxSize)`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

PurgeSpace

Determines the total amount of free memory and the size of the largest allocatable block in the current heap zone if it were purged. (Deprecated in Mac OS X v10.4. There is no replacement; heaps are never purged in Mac OS X.)

```
void PurgeSpace (
   long *total,
   long *contig
);
```

Parameters

total

On return, the total amount of free memory, in bytes, in the current heap zone if it were purged. This amount includes space that is already free.

contig

On return, the size of the largest contiguous block of free memory in the current heap zone if it were purged.

Discussion
The `PurgeSpace` function does not actually purge the current heap zone.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

PurgeSpaceContiguous

(Deprecated in Mac OS X v10.4. There is no replacement; heaps are never purged in Mac OS X.)

```
long PurgeSpaceContiguous (
   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
`MacMemory.h`

PurgeSpaceTotal

(Deprecated in Mac OS X v10.4. There is no replacement; heaps are never purged in Mac OS X.)

```
long PurgeSpaceTotal (
   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
`MacMemory.h`

ReallocateHandle

Allocates a new relocatable block of a specified size and sets a handle's master pointer to point to the new block.

```
void ReallocateHandle (
   Handle h,
   Size byteCount
);
```

Parameters

h

A handle to a relocatable block.

byteCount

The desired new logical size (in bytes) of the relocatable block. The new block is unlocked and unpurgeable.

Discussion
Usually you use `ReallocateHandle` to reallocate space for a block that you have emptied or the Memory Manager has purged. If the handle references an existing block, `ReallocateHandle` releases that block before creating a new one.

If many handles reference a single purged, relocatable block, you need to call `ReallocateHandle` on just one of them.

To reallocate space for a resource that has been purged, you should call `LoadResource`, not `ReallocateHandle`. To resize relocatable blocks, you should call the `SetHandleSize` (page 1425) function.

Currently in Mac OS 8 and 9, the `ReallocateHandle` function releases any existing relocatable block referenced by the handle `h` before allocating a new one. This behavior means that if an error occurs when calling `ReallocateHandle`, the handle `h` will be set to `NULL`. This behavior does not occur in the Mac OS X implementation.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Because `ReallocateHandle` might purge and allocate memory, you should not call it at interrupt time.

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

Declared In
`MacMemory.h`

RecoverHandle

Returns a handle to a relocatable block pointed to by a specified `pointer`.

```
Handle RecoverHandle (
   Ptr p
);
```

Parameters

p
The master pointer to a relocatable block.

Return Value
A handle to a relocatable block point to by p. If `p` does not point to a valid block, the results of `RecoverHandle` are undefined.

Discussion
The Memory Manager does not allow you to change relocatable blocks into nonrelocatable blocks, or vice-versa. However, if you no longer have access to a handle but still have access to its master pointer p, you can use the `RecoverHandle` function to recreate a handle to the relocatable block referenced by p.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Even though `RecoverHandle` does not move or purge memory, you should not call it at interrupt time.

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

Declared In
`MacMemory.h`

ReleaseMemoryData

Releases the data of a portion of the address space. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
OSErr ReleaseMemoryData (
   void *address,
   unsigned long count
);
```

Return Value
This function always returns a value of `noErr`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

ReserveMem

Reserves space for a block of memory as close to the bottom of the current heap zone as possible. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
void ReserveMem (
   Size cbNeeded
);
```

Parameters

cbNeeded
The number of bytes to reserve near the bottom of the heap.

Discussion
The `ReserveMem` function attempts to create free space for the specified number of contiguous logical bytes at the lowest possible position in the current heap zone. It pursues every available means of placing the block as close as possible to the bottom of the zone, including moving other relocatable blocks upward, expanding the zone (if possible), and purging blocks from it.

Use the `ReserveMem` function when allocating a relocatable block that you intend to lock for long periods of time. This helps prevent heap fragmentation because it reserves space for the block as close to the bottom of the heap as possible. Consistent use of `ReserveMem` for this purpose ensures that all locked, relocatable blocks and nonrelocatable blocks are together at the bottom of the heap zone and thus do not prevent unlocked relocatable blocks from moving about the zone.

Because `ReserveMem` does not actually allocate the block, you must combine calls to `ReserveMem` with calls to the `NewHandle` function.

Do not use the `ReserveMem` function for a relocatable block you intend to lock for only a short period of time. If you do so and then allocate a nonrelocatable block above it, the relocatable block becomes trapped under the nonrelocatable block when you unlock that relocatable block.

It isn't necessary to call `ReserveMem` to reserve space for a nonrelocatable block, because the `NewPtr` function calls it automatically.

Also, you do not need to call `ReserveMem` to reserve memory before you load a locked resource into memory, because the Resource Manager calls `ReserveMem` automatically.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Because the `ReserveMem` function could move and purge memory, you should not 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
`MacMemory.h`

SetGrowZone

Specifies the current heap zone's grow-zone function. (Deprecated in Mac OS X v10.4. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never called.)

```
void SetGrowZone (
   GrowZoneUPP growZone
);
```

Parameters

growZone

A pointer to the grow-zone function. A `NULL` value removes any previous grow-zone function from the zone.

Discussion
To specify a grow-zone function for the current heap zone, pass a pointer to that function to the `SetGrowZone` function. Usually you call this function early in the execution of your application.

If you initialize your own heap zones besides the application and system zones, you can alternatively specify a grow-zone function as a parameter to the `InitZone` function.

The Memory Manager calls the grow-zone function only after exhausting all other avenues of satisfying a memory request, including compacting the zone, increasing its size (if it is the original application zone and is not yet at its maximum size), and purging blocks from it.

See "Grow-Zone Operations" for a complete description of a grow-zone 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
`MacMemory.h`

SetHandleSize

Changes the logical size of the relocatable block corresponding to the specified handle.

```
void SetHandleSize (
   Handle h,
   Size newSize
);
```

Parameters

h

A handle to a relocatable block.

newSize

The desired new logical size, in bytes, of the relocatable block.

Discussion

`SetHandleSize` tries to change the size of the allocation to *newSize*. If there is not enough room to enlarge the memory allocation pointed to by *h*, `SetHandleSize` creates a new allocation, copies as much of the old data pointed to by *h* as will fit to the new allocation, and frees the old allocation. `SetHandleSize` might need to move the relocatable block to obtain enough space for the resized block. Thus, for best results you should unlock a block before resizing it.

An attempt to increase the size of a locked block might fail, because of blocks above and below it that are either nonrelocatable or locked. You should be prepared for this possibility.

Instead of using the `SetHandleSize` function to set the size of a handle to `0`, you can use the `EmptyHandle` (page 1392) function.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Because `SetHandleSize` allocates memory, you should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

SoftVDigX

Declared In

`MacMemory.h`

SetPtrSize

Changes the logical size of the nonrelocatable block corresponding to a pointer.

```
void SetPtrSize (
   Ptr p,
   Size newSize
);
```

Parameters

p

A pointer to a nonrelocatable block.

newSize

The desired new logical size, in bytes, of the nonrelocatable block.

Discussion
An attempt to increase the size of a nonrelocatable block might fail because of a block above it that is either nonrelocatable or locked. You should be prepared for this possibility.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Because `SetPtrSize` allocates memory, you should not call it at interrupt time.

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

Declared In
`MacMemory.h`

StackSpace

Returns the amount of space between the bottom of the stack and the top of the application heap. (Deprecated in Mac OS X v10.5. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
long StackSpace (
   void
);
```

Return Value
The current amount of stack space, in bytes, between the current stack pointer and the application heap.

Discussion
Usually you determine the maximum amount of stack space needed before you ship your application. Thus this function is generally useful only during debugging to determine how big to make the stack. However, if your application calls a recursive function that conceivably could call itself many times, that function should keep track of the stack space and take appropriate action if it becomes too low.

Call the function `MemError` (page 1410) to get the result code. See "Memory Manager Result Codes" (page 1443).

Special Considerations

StackSpace must not be called at interrupt time, as it may alter location MemErr.

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
`MacMemory.h`

TempDisposeHandle

Releases a relocatable block in the temporary heap. (Deprecated in Mac OS X v10.5. Use `DisposeHandle` (page 1390) instead; Mac OS X does not have a separate temporary memory heap.)

```
void TempDisposeHandle (
   Handle h,
   OSErr *resultCode
);
```

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
`MacMemory.h`

TempFreeMem

Returns the maximum amount of free memory in the temporary heap. (Deprecated in Mac OS X v10.4. There is no replacement function; Mac OS X does not have a separate temporary memory heap.)

```
long TempFreeMem (
   void
);
```

Return Value
The total amount of free temporary memory, in bytes, that you could allocate by calling `TempNewHandle`. Because these bytes might be dispersed throughout memory, it is ordinarily not possible to allocate a single relocatable block of that size.

Discussion
Returns the total amount of memory available for temporary allocation.

Special Considerations

In Mac OS X, there is no separate temporary memory heap. This function always returns a large value, because virtual memory is always available to fulfill any request for memory. You can assume that any reasonable memory allocation request will succeed.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

TempHLock

Locks a relocatable block in the temporary heap. (Deprecated in Mac OS X v10.4. Use `HLock` (page 1398) instead; Mac OS X does not have a separate temporary memory heap.)

```
void TempHLock (
   Handle h,
   OSErr *resultCode
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
MacMemory.h

TempHUnlock

Unlocks a relocatable block in the temporary heap. (Deprecated in Mac OS X v10.4. Use `HUnlock` (page 1403) instead; Mac OS X does not have a separate temporary memory heap.)

```
void TempHUnlock (
   Handle h,
   OSErr *resultCode
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
MacMemory.h

TempMaxMem

Returns the maximum amount of temporary memory available. (Deprecated in Mac OS X v10.4. There is no replacement function; Mac OS X does not have a separate temporary memory heap.)

```
Size TempMaxMem (
   Size *grow
);
```

Parameters

grow

On return, this parameter always contains 0 after the function call because temporary memory does not come from the application's heap zone, and only that zone can grow. Ignore this parameter.

Return Value
The size of the largest contiguous block available for temporary allocation.

Discussion
Compacts the current heap zone and returns the size of the largest contiguous block available for temporary allocation.

Special Considerations

In Mac OS X, there is no separate temporary memory heap. This function always returns a large value, because virtual memory is always available to fulfill any request for memory. You can assume that any reasonable memory allocation request will succeed.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

TempNewHandle

Allocates a new relocatable block of temporary memory.

```
Handle TempNewHandle (
   Size logicalSize,
   OSErr *resultCode
);
```

Parameters

`logicalSize`

The requested logical size, in bytes, of the new temporary block of memory.

`resultCode`

On return, the result code from the function call. See "Memory Manager Result Codes" (page 1443).

Return Value

A handle to a block of size `logicalSize`. If it cannot allocate a block of that size, the function returns `NULL`.

Discussion

Before calling `TempNewHandle`, you should call `TempFreeMem` or `TempMaxMem` to make sure that there is enough free space to satisfy the request.

Because `TempNewHandle` might allocate memory, you should not call it at interrupt time.

Carbon Porting Notes

Temporary memory allocations will actually come from the applications's address space in Mac OS X. However, Carbon applications running under Mac OS 8.x will be able to get true temporary memory.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacMemory.h`

TempTopMem

Returns the location of the top of the temporary heap. (Deprecated in Mac OS X v10.4. There is no replacement function; Mac OS X does not have a separate temporary memory heap.)

```
Ptr TempTopMem (
   void
);
```

Discussion
In Mac OS X, this function always returns `NULL`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

TopMem

Returns a pointer to the byte at the top of an application's partition. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
Ptr TopMem (
   void
);
```

Discussion
Deprecated. Refer to `MacMemory.h` for information on replacement functions.

`TopMem` obtains a pointer to the byte at the top of an application's partition, directly above the jump table. `TopMem` does this to maintain compatibility with programs that check `TopMem` to find out how much memory is installed in a computer. The preferred method of obtaining this information is with the `Gestalt` function.

The function exhibits special behavior at startup time, and the value it returns controls the amount by which an extension can lower the value of the global variable `BufPtr` at startup time. If you are writing a system extension, you should not lower the value of `BufPtr` by more than `MemTop DIV 2 + 1024`. If you do lower `BufPtr` too far, the startup process generates an out-of-memory system error.

You should never need to call `TopMem` except during the startup process.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

UnholdMemory

Makes a currently held range of memory eligible for paging again. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
OSErr UnholdMemory (
   void *address,
   unsigned long count
);
```

Parameters

address

A pointer indicating the starting address of the range of memory to be released.

count

The size, in bytes, of the range of memory to be released.

Return Value

This function always returns a value of `noErr`.

Discussion

If the starting address you supply to the `UnholdMemory` function is not on a page boundary, then `UnholdMemory` rounds down to the nearest page boundary. Similarly, if the specified range does not end on a page boundary, `UnholdMemory` rounds up the value you pass in the `count` parameter so that the entire range of memory is released.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

Callbacks

All Memory Manager callbacks are deprecated in Mac OS X. They are non-functional.

GrowZoneProcPtr

Deprecated.

```
typedef long (*GrowZoneProcPtr) (
    Size cbNeeded
);
```

If you name your function `MyGrowZoneProc`, you would declare it like this:

```
long MyGrowZoneProc (
    Size cbNeeded
);
```

Parameters

cbNeeded

The physical size, in bytes, of the needed block, including the block header. The grow-zone function should attempt to create a free block of at least this size.

Return Value
The number of bytes of memory the function has freed.

Discussion
User-defined function that creates free space in the heap.

Whenever the Memory Manager has exhausted all available means of creating space within your application heap—including purging, compacting, and (if possible) expanding the heap—it calls your application-defined grow-zone function. The grow-zone function can do whatever is necessary to create free space in the heap. Typically, a grow-zone function marks some unneeded blocks as purgeable or releases an emergency memory reserve maintained by your application.

The grow-zone function should return a nonzero value equal to the number of bytes of memory it has freed, or zero if it is unable to free any. When the function returns a nonzero value, the Memory Manager once again purges and compacts the heap zone and tries to reallocate memory. If there is still insufficient memory, the Memory Manager calls the grow-zone function again (but only if the function returned a nonzero value the previous time it was called). This mechanism allows your grow-zone function to release just a little bit of memory at a time. If the amount it releases at any time is not enough, the Memory Manager calls it again and gives it the opportunity to take more drastic measures.

The Memory Manager might designate a particular relocatable block in the heap as protected; your grow-zone function should not move or purge that block. You can determine which block, if any, the Memory Manager has protected by calling the `GZSaveHnd` function in your grow-zone function.

Remember that the Memory Manager calls a grow-zone function while attempting to allocate memory. As a result, your grow-zone function should not allocate memory itself or perform any other actions that might indirectly cause memory allocation (such as calling functions in unloaded code segments or displaying dialog boxes).

You install a grow-zone function by passing its address to the `InitZone` function when you create a new heap zone or by calling the `SetGrowZone` function at any other time.

Your grow-zone function might be called at a time when the system is attempting to allocate memory and the value in the A5 register is not correct. If your function accesses your application's A5 world or makes any trap calls, you need to set up and later restore the A5 register by calling `SetCurrentA5` and `SetA5`. See the chapter "Memory Management Utilities" in this book for a description of these two functions.

Because of the optimizations performed by some compilers, the actual work of the grow-zone function and the setting and restoring of the A5 register might have to be placed in separate functions.

See the chapter "Introduction to Memory Management" for a definition of a sample grow-zone function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

PurgeProcPtr

Deprecated.

```
typedef void (*PurgeProcPtr) (
    Handle blockToPurge
);
```

If you name your function `MyPurgeProc`, you would declare it like this:

```
void MyPurgeProc (
    Handle blockToPurge
);
```

Parameters

blockToPurge

A handle to the block that is about to be purged.

Discussion

User-defined function called when the Memory Manager needs to purge a block or allocate memory.

Whenever the Memory Manager needs to purge a block from the application heap, it first calls any application-defined purge-warning function that you have installed. The purge-warning function can, if necessary, save the contents of that block or otherwise respond to the warning.

Your purge-warning function is called during a memory-allocation request. As a result, you should not call any functions that might cause memory to be moved or purged. In particular, if you save the data of the block in a file, the file should already be open when your purge-warning function is called, and you should write the data synchronously.

You should not dispose of or change the purgeable status of the block whose handle is passed to your function.

To install a purge-warning function, you need to assign its address to the `purgeProc` field of the associated zone header.

Note that if you call the Resource Manager function `SetResPurge` with the parameter `TRUE`, any existing purge-warning function is replaced by a purge-warning function installed by the Resource Manager. You can execute both warning functions by calling `SetResPurge`, saving the existing value of the `purgeProc` field of the zone header, and then reinstalling your purge-warning function. Your purge-warning function should call the Resource Manager's purge-warning function internally.

Your purge-warning function might be called at a time when the system is attempting to allocate memory and the value in the A5 register is not correct. If your function accesses your application's A5 world or makes any trap calls, you need to set up and later restore the A5 register by calling `SetCurrentA5` and `SetA5`.

Because of the optimizations performed by some compilers, the actual work of the purge-warning function and the setting and restoring of the A5 register might have to be placed in separate functions.

The Memory Manager calls your purge-warning function for every handle that is about to be purged (not necessarily for every purgeable handle in your heap, however). Your function should be able to determine quickly whether the handle that the Memory Manager is about to purge points to data you need to save or otherwise process.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
MacMemory.h

UserFnProcPtr

Deprecated.

```
typedef void (*UserFnProcPtr) (
    void *parameter
);
```

If you name your function `MyUserFnProc`, you would declare it like this:

```
void MyUserFnProc (
    void * parameter
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacMemory.h

Data Types

All Memory Manager data types are deprecated in Mac OS X. They are not used.

BackingFileID

Deprecated.

```
typedef struct * BackingFileID;
```

Special Considerations

FileViewAccess

Deprecated.

```
typedef UInt32 FileViewAccess;
enum {
    kFileViewAccessReadBit = 0,
    kFileViewAccessWriteBit = 1,
    kFileViewAccessExecuteBit = 2,
    kFileViewAccessReadMask = 1,
    kFileViewAccessWriteMask = 2,
    kFileViewAccessExecuteMask = 4,
    kFileViewAccessExcluded = 0,
    kFileViewAccessReadOnly = 5,
    kFileViewAccessReadWrite = 7
};
```

Special Considerations

FileViewID

Deprecated.

```
typedef struct * FileViewID;
```

Special Considerations

FileViewInformation

Deprecated.

```
struct FileViewInformation {
    ProcessSerialNumber owningProcess;
    LogicalAddress viewBase;
    ByteCount viewLength;
    BackingFileID backingFile;
    UInt64 backingBase;
    FileViewAccess access;
    ByteCount guardLength;
    FileViewOptions options;
};
```

FileViewOptions

Deprecated.

```
typedef OptionBits FileViewOptions;
```

GrowZoneUPP

Deprecated.

```
typedef GrowZoneProcPtr GrowZoneUPP;
```

Discussion
For more information, see the description of the GrowZoneUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

LogicalToPhysicalTable

Deprecated.

```
struct LogicalToPhysicalTable {
    MemoryBlock logical;
    MemoryBlock physical[8];
};
typedef struct LogicalToPhysicalTable LogicalToPhysicalTable;
```

Fields

`logical`
A logical block of memory whose corresponding physical blocks are to be determined.

`physical`
A physical translation table that identifies the blocks of physical memory corresponding to the logical block identified in the `logical` field.

Discussion
The `GetPhysical` function uses a translation table to hold information about a logical address range and its corresponding physical addresses. A translation table is defined by the data type LogicalToPhysicalTable.

Special Considerations

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

MappedFileAttributes

Deprecated.

```
typedef UInt32 MappedFileAttributes;
enum {
    kIsMappedScratchFile = 1
};
```

MappedFileInformation

Deprecated.

```
struct MappedFileInformation {
    ProcessSerialNumber owningProcess;
    FSRef *ref;
    HFSUniStr255 *forkName;
    MappingPrivileges privileges;
    UInt64 currentSize;
    MappedFileAttributes attributes;
};
```

MappingPrivileges

Deprecated.

```
typedef UInt32 MappingPrivileges;
enum {
    kInvalidMappedPrivileges = 0,
    kCanReadMappedFile = 1,
    kCanWriteMappedFile = 2,
    kNoProcessMappedFile = -2147483648,
    kValidMappingPrivilegesMask = -2147483645
};
```

Special Considerations

MemoryBlock

Deprecated.

```
struct MemoryBlock {
    void * address;
    unsigned long count;
};
typedef struct MemoryBlock MemoryBlock;
```

Fields

`address`
A pointer to the beginning of a block of memory.

`count`
The number of bytes in the block of memory.

Discussion

The `GetPhysical` function uses a structure of type MemoryBlock to hold information about a block of memory, either logical or physical.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

PurgeUPP

Deprecated.

```
typedef PurgeProcPtr PurgeUPP;
```

Discussion
For more information, see the description of the PurgeUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

StatusRegisterContents

Deprecated.

```
typedef  StatusRegisterContents;
```

Special Considerations

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

UserFnUPP

Deprecated.

```
typedef UserFnProcPtr UserFnUPP;
```

Discussion
For more information, see the description of the UserFnUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

VolumeVirtualMemoryInfo

Deprecated.

```
struct VolumeVirtualMemoryInfo {
    PBVersion version;
    SInt16 volumeRefNum;
    Boolean inUse;
    UInt8 _fill;
    UInt32 vmOptions;
};
typedef struct VolumeVirtualMemoryInfo VolumeVirtualMemoryInfo;
typedef VolumeVirtualMemoryInfo * VolumeVirtualMemoryInfoPtr;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

Zone

Deprecated.

```
struct Zone {
    Ptr bkLim;
    Ptr purgePtr;
    Ptr hFstFree;
    long zcbFree;
    GrowZoneUPP gzProc;
    short moreMast;
    short flags;
    short cntRel;
    short maxRel;
    short cntNRel;
    SInt8 heapType;
    SInt8 unused;
    short cntEmpty;
    short cntHandles;
    long minCBFree;
    PurgeUPP purgeProc;
    Ptr sparePtr;
    Ptr allocPtr;
    short heapData;
};
typedef struct Zone Zone;
typedef Zone * THz;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

Constants

All Memory Manager constants are deprecated in Mac OS X. They are not used.

Default Physical Entry Count Constant

Deprecated.

```
enum {
    defaultPhysicalEntryCount = 8
};
```

Discussion
The defaultPhysicalEntryCount constant represents the default number of physical blocks in a table.

k32BitHeap

Deprecated.

```
enum {
    k32BitHeap = 1,
    kNewStyleHeap = 2,
    kNewDebugHeap = 4
};
```

kFileViewInformationVersion1

Deprecated.

```
enum {
    kFileViewInformationVersion1 = 1
};
```

kHandleIsResourceBit

Deprecated.

```
enum {
    kHandleIsResourceBit = 5,
    kHandlePurgeableBit = 6,
    kHandleLockedBit = 7
};
```

kHandleIsResourceMask

Deprecated.

```
enum {
    kHandleIsResourceMask = 0x20,
    kHandlePurgeableMask = 0x40,
    kHandleLockedMask = 0x80
};
```

kMapEntireFork

Deprecated.

```
enum {
    kMapEntireFork = -1
};
```

Constants
`kMapEntireFork`

kMappedFileInformationVersion1

Deprecated.

```
enum {
    kMappedFileInformationVersion1 = 1
};
```

kPageInMemory

Deprecated.

```
typedef short PageState;
enum {
    kPageInMemory = 0,
    kPageOnDisk = 1,
    kNotPaged = 2
};
```

Discussion
The `GetPageState` function obtains the state value of a page of logical memory. The `PageState` data type defines constants that represent these possible state values.

Debuggers need a way to display the contents of memory without paging or to display the contents of pages currently on disk. The `GetPageState` function obtains a constant from the PageState data type to specify the state of a page containing a virtual address. A debugger can use this information to determine whether certain memory addresses should be referenced. Note that ROM and I/O space are not pageable and therefore are considered not paged.

kVolumeVirtualMemoryInfoVersion1

Deprecated.

```
enum {
    kVolumeVirtualMemoryInfoVersion1 = 1
};
```

maxSize

Deprecated.

```
enum {
    maxSize = 0x7FFFFFF0
};
```

Result Codes

The most common result codes returned by the Memory Manager are listed below.

Result Code	Value	Description
`menuPrgErr`	84	A menu was purged. Available in Mac OS X v10.0 and later.
`negZcbFreeErr`	33	A heap has been corrupted. Available in Mac OS X v10.0 and later.
`memROZErr`	-99	Operation on a read-only zone. This result code is not relevant in Mac OS X. Available in Mac OS X v10.0 and later.
`memFullErr`	-108	Not enough memory in heap. Available in Mac OS X v10.0 and later.
`nilHandleErr`	-109	Handle argument is `NULL`. Available in Mac OS X v10.0 and later.
`memAdrErr`	-110	Address is odd or out of range. Available in Mac OS X v10.0 and later.
`memWZErr`	-111	Attempt to operate on a free block. Available in Mac OS X v10.0 and later.
`memPurErr`	-112	Attempt to purge a locked or unpurgeable block. Available in Mac OS X v10.0 and later.
`memAZErr`	-113	Address in zone check failed. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
memPCErr	-114	Pointer check failed. Available in Mac OS X v10.0 and later.
memBCErr	-115	Block check failed. Available in Mac OS X v10.0 and later.
memSCErr	-116	Size check failed. Available in Mac OS X v10.0 and later.
memLockedErr	-117	Block is locked. Available in Mac OS X v10.0 and later.

Mixed Mode Manager Reference

Framework: CoreServices/CoreServices.h

Declared in MixedMode.h

Overview

Mac OS X does not require the Mixed Mode Manager, and does not support its functions. These unsupported functions are listed in the Appendix. The functions have been removed from the Mixed Mode Manager and redefined as macros for the purpose of source compatibility with code ported to CFM. See the header file MixedMode.h for details on these macros and their usage.

You do not need to remove Mixed Mode Manager calls from your application for compatibility with Mac OS X, and may want to retain them for source code compatibility with previous versions of the Mac OS.

The Mixed Mode Manager managed the mixed-mode architecture of PowerPC processor-based computers running 680x0-based code (including system software, applications, and stand-alone code modules). The Mixed Mode Manager cooperated with the 68LC040 Emulator to provide a fast, efficient, and virtually transparent method for code in one instruction set architecture to call code in another architecture. The Mixed Mode Manager handled all the details of switching between architectures.

The Mixed Mode Manager was intended to operate transparently to most applications and other software.

Although Mac OS X does not run 68K code, Carbon supports universal procedure pointers (UPPs) transparently, so you do not have to change them or remove them from your code. You may want to keep Mixed Mode Manager calls in your application to maintain source code compatibility with the previous versions of the Mac OS. Mixed Mode Manager calls from Carbon applications running on Mac OS 8 or 9 will function normally.

The Mixed Mode Manager was used by developers who

- wanted to recompile their applications into PowerPC code and their applications passed the address of some routines to the Mac OS using a reference of type `ProcPtr`
- created applications–written in either PowerPC or 680x0 code–that support installable code modules that might be written in a different architecture
- wrote stand-alone code (for example, a VBL task or a component) that could be called from either the PowerPC native environment or the 680x0 emulated environment
- wrote debuggers or other software that needed to know about the structure of the stack at any time (for example, during a mode switch)

Mac OS X will not run 68K code. Although Carbon supports universal procedure pointers (UPPs), applications should use ProcPtrs for their own code and plug-ins and use the new system-supplied UPP creation functions for Toolbox callback UPPs. You still need to dispose of those UPPs (using the corresponding disposal function), so that any allocated memory can be cleaned up when your application is running on Mac OS 8 or 9.

Data Types

MixedModeStateRecord

Contains mixed mode state information.

```
struct MixedModeStateRecord {
    UInt32 state1;
    UInt32 state2;
    UInt32 state3;
    UInt32 state4;
};
typedef struct MixedModeStateRecord MixedModeStateRecord;
```

Fields

`state1`

`state2`

`state3`

`state4`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MixedMode.h`

ProcInfoType

Defines a data type used to encode a routine's procedure information.

```
typedef unsigned long ProcInfoType;
```

Discussion

The Mixed Mode Manager uses a long word of type `ProcInfoType` to encode a routine's procedure information, which contains essential information about the calling conventions and other features of a routine. These values specify

- the routine's calling conventions
- the sizes and locations of the routine's parameters, if any
- the size and location of the routine's result, if any

The Mixed Mode Manager provides a number of constants that you can use to specify the procedure information. See "Procedure Information Size Constants" (page 1455) , "ProcInfo Field Offset And Width Constants" (page 1456) , "Calling Convention Constants" (page 1450) , "Special Case Calling Convention Constants" (page 1464) , and "Register Constants" (page 1459).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MixedMode.h`

RDFlagsType

Defines a data type for routine descriptor flags.

```
typedef UInt8 RDFlagsType;
```

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

Declared In
`MixedMode.h`

RoutineDescriptor

Contains information used by the Mixed Mode Manager to execute a routine.

```
struct RoutineDescriptor {
    UInt16 goMixedModeTrap;
    SInt8 version;
    RDFlagsType routineDescriptorFlags;
    UInt32 reserved1;
    UInt8 reserved2;
    UInt8 selectorInfo;
    UInt16 routineCount;
    RoutineRecord routineRecords[1];
};
typedef struct RoutineDescriptor RoutineDescriptor;
typedef RoutineDescriptor * RoutineDescriptorPtr;
typedef RoutineDescriptorPtr RoutineDescriptorHandle;
```

Fields

`goMixedModeTrap`
: An A-line instruction that is used privately by the Mixed Mode Manager. When the emulator encounters this instruction, it transfers control to the Mixed Mode Manager. This field contains the value $AAFE.

`version`
: The version number of the `RoutineDescriptor` data type.

`routineDescriptorFlags`
: A set of routine descriptor flags. Currently, all the bits in this field should be set to 0, unless you are specifying a routine descriptor for a dispatched routine.

`reserved1`
: Reserved. This field must initially be 0.

`reserved2`
: Reserved. This field must be 0.

`selectorInfo`
: Reserved. This field must be 0.

`routineCount`
: The index of the final routine record in the following array, of routineRecords. Because the `routineRecords` array is zero-based, this field does not contain an actual count of the routine records contained in that array. Often, you will use a routine descriptor to describe a single procedure, in which case this field should contain the value 0. You can, however, construct a routine descriptor that contains pointers to both 680x0 and PowerPC code (known as a “fat” routine descriptor). In that case, this field should contain the value 1.

`routineRecords`
: An array of routine records for the routines described by this routine descriptor. See "`RoutineRecord`" (page 1448) for the structure of a routine record. This array is zero-based.

Discussion
A routine descriptor is a data structure used by the Mixed Mode Manager to execute a routine. The external interface to a routine descriptor is through a universal procedure pointer, of type `UniversalProcPtr`, which is defined as a procedure pointer (if the code is 680x0 code) or as a pointer to a routine descriptor (if the code is PowerPC code). A routine descriptor is defined by the `RoutineDescriptor` data type.

Your application (or other software) should never attempt to guide its execution by inspecting the value in the `ISA` field of a routine record and jumping to the address in the `procDescriptor` field.

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

Declared In
`MixedMode.h`

RoutineFlagsType

Defines a data type for routine flags.

```
typedef unsigned short RoutineFlagsType;
```

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

Declared In
`MixedMode.h`

RoutineRecord

Cntains information about a particular routine.

```
struct RoutineRecord {
    ProcInfoType procInfo;
    SInt8 reserved1;
    ISAType ISA;
    RoutineFlagsType routineFlags;
    ProcPtr procDescriptor;
    UInt32 reserved2;
    UInt32 selector;
};
typedef struct RoutineRecord RoutineRecord;
typedef RoutineRecord * RoutineRecordPtr;
typedef RoutineRecordPtr RoutineRecordHandle;
```

Fields

`procInfo`

A value of type `ProcInfoType` that encodes essential information about the routine's calling conventions and parameters. See "Procedure Information Size Constants" (page 1455), "ProcInfo Field Offset And Width Constants" (page 1456), "Calling Convention Constants" (page 1450), "Special Case Calling Convention Constants" (page 1464), and "Register Constants" (page 1459) for descriptions of the constants you can use to set this field.

`reserved1`

Reserved. This field must be 0.

`ISA`

The instruction set architecture of the routine. See "Instruction Set Architectures" (page 1452) for a complete listing of the constants you can use to set this field.

`routineFlags`

A value of type `RoutineFlagsType` that contains a set of flags describing the routine. See "Routine Entry Point Flags" (page 1463), "Fragment Flags" (page 1452), "ISA Flags" (page 1453), "Routine Selector Flags" (page 1463), and "Default Routine Flags" (page 1451) for descriptions of the constants you can use to set this field.

`procDescriptor`

A pointer to the routine's code. If the routine consists of 680x0 code and the `kProcDescriptorIsAbsolute` flag is set in the `routineFlags` field, then this field contains the address of the routine's entry point. If the routine consists of 680x0 code and the `kProcDescriptorIsRelative` flag is set, then this field contains the offset from the beginning of the routine descriptor to the routine's entry point. If the routine consists of PowerPC code, the `kFragmentIsPrepared` flag is set, and the `kProcDescriptorIsAbsolute` flag is set, then this field contains the address of the routine's transition vector. If the routine consists of PowerPC code, the `kFragmentNeedsPreparing` flag is set, and the `kProcDescriptorIsRelative` flag is set, then this field contains the offset from the beginning of the routine descriptor to the routine's entry point.

`reserved2`

Reserved. This field must be 0.

`selector`

Reserved. This field must be 0. For routines that are dispatched, this field contains the routine selector.

Discussion

A routine record is a data structure that contains information about a particular routine. The routine descriptor specifies, among other things, the instruction set architecture of the routine, the number and size of the routine's parameters, the routine's calling conventions, and the routine's location in memory. At least one routine record is contained in the `routineRecords` field of a routine descriptor. A routine record is defined by the `RoutineRecord` data type.

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

Declared In
`MixedMode.h`

Constants

Calling Convention Constants

Specify a routine's calling conventions.

```
typedef unsigned short CallingConventionType;
enum {
    kPascalStackBased = 0,
    kCStackBased = 1,
    kRegisterBased = 2,
    kD0DispatchedPascalStackBased = 8,
    kD1DispatchedPascalStackBased = 12,
    kD0DispatchedCStackBased = 9,
    kStackDispatchedPascalStackBased = 14,
    kThinkCStackBased = 5
};
```

Constants

`kPascalStackBased`
The routine follows normal Pascal calling conventions.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kCStackBased`
The routine follows the C calling conventions employed by the MPW development environment.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterBased`
The parameters are passed in registers.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kD0DispatchedPascalStackBased`
The parameters are passed on the stack according to Pascal conventions, and the routine selector is passed in register D0.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kD1DispatchedPascalStackBased`
The parameters are passed on the stack according to Pascal conventions, and the routine selector is passed in register D1.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kD0DispatchedCStackBased`
The parameters are passed on the stack according to C conventions, and the routine selector is passed in register D0.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kStackDispatchedPascalStackBased`
The routine selector and the parameters are passed on the stack.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kThinkCStackBased`
The routine follows the C calling conventions employed by the THINK C software development environment. Arguments are passed on the stack from right to left, and a result is returned in register D0. All arguments occupy an even number of bytes on the stack. An argument having the size of a `char` is passed in the high-order byte. You should always provide function prototypes; failure to do so may cause THINK C to generate code that is incompatible with this parameter-passing convention.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion
These constants are used by the `ProcInfoType` (page 1446) type to specify a routine's calling conventions.

Default Routine Flags

Specify defaults for a routine.

```
enum {
    kRoutineIsNotDispatchedDefaultRoutine = 0x00,
    kRoutineIsDispatchedDefaultRoutine = 0x10
};
```

Constants
`kRoutineIsNotDispatchedDefaultRoutine`
This routine is not the default routine for a set of routines that is dispatched using a routine selector.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRoutineIsDispatchedDefaultRoutine`
This routine is the default routine for a set of routines that is dispatched using a routine selector. If a set of routines is dispatched using a routine selector and the routine corresponding to a specified selector cannot be found, this default routine is called. This routine must be able to accept the same procedure information for all routines. If possible, it is passed the procedure information passed in a call to `CallUniversalProc`.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion
The `routineFlags` field of a routine record contains a set of flags that specify information about a routine. You can use constants to specify the desired routine flags. Currently, only 5 of the 16 bits in a routine flags word are defined. You should set all the other bits to 0.

In general, you should use the constant `kRoutineIsNotDispatchedDefaultRoutine`. The constant and `kRoutineIsDispatchedDefaultRoutine` is reserved for use with selector-based system software routines.

Fragment Flags

Used in the `routineFlags` field of a routine record.

```
enum {
    kFragmentIsPrepared = 0x00,
    kFragmentNeedsPreparing = 0x02
};
```

Constants

`kFragmentIsPrepared`

The fragment containing the code to be executed is already loaded into memory and prepared by the Code Fragment Manager.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kFragmentNeedsPreparing`

The fragment containing the code to be executed needs to be loaded into memory and prepared by the Code Fragment Manager. If this flag is set, the `kPowerPCISA` and `kProcDescriptorIsRelative` flags should also be set.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion

The `routineFlags` field of a routine record contains a set of flags that specify information about a routine. You can use constants to specify the desired routine flags. Currently, only 5 of the 16 bits in a routine flags word are defined. You should set all the other bits to 0.

Instruction Set Architectures

Used in the `ISA` field of a routine record.

```
typedef SInt8 ISAType;
enum {
    kM68kISA = 0,
    kPowerPCISA = 1
};
```

Constants

`kM68kISA`

The routine consists of Motorola 680x0 code.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kPowerPCISA`

The routine consists of PowerPC code.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion
The `ISA` field of a routine record contains a flag that specifies the instruction set architecture of a routine. You can use constants to specify the instruction set architecture.

ISA Flags

Used in the `routineFlags` field of a routine record.

```
enum {
    kUseCurrentISA = 0x00,
    kUseNativeISA = 0x04
};
```

Constants

`kUseCurrentISA`
If possible, use the current instruction set architecture when executing a routine.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kUseNativeISA`
Use the native instruction set architecture when executing a routine.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

Discussion
The `routineFlags` field of a routine record contains a set of flags that specify information about a routine. You can use constants to specify the desired routine flags. Currently, only 5 of the 16 bits in a routine flags word are defined. You should set all the other bits to 0.

Current Mixed Mode State

Specifies the current version of the mixed-mode state record.

```
enum {
    kCurrentMixedModeStateRecord = 1
};
```

RTA Types

```
typedef SInt8 RTAType;
enum {
    kOld68kRTA = 0 << 4,
    kPowerPCRTA = 0 << 4,
    kCFM68kRTA = 1 << 4
};
```

Constants

`kOld68kRTA`
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kPowerPCRTA`
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kCFM68kRTA`
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

Procedure Descriptors

```
enum {
    kProcDescriptorIsProcPtr = 0x00,
    kProcDescriptorIsIndex = 0x20
};
```

Constants

`kProcDescriptorIsProcPtr`
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kProcDescriptorIsIndex`
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

Routine Descriptor Version

Specifies the version of routine descriptor.

```
enum {
    kRoutineDescriptorVersion = 7
};
```

Special Case Constant

Used to specify a special case.

```
enum {
    kSpecialCase = 0x000F
};
```

Constants

`kSpecialCase`
The routine is a special case. You can use the following constants to specify a special case.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

kX86ISA

```
enum {
    kX86ISA = 2
};
```

Constants

`kX86ISA`

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

kX86RTA

```
enum {
    kX86RTA = 2 << 4
};
```

Constants

`kX86RTA`

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

_MixedModeMagic

```
enum {
    _MixedModeMagic = 0xAAFE
};
```

Constants

`_MixedModeMagic`

Procedure Information Size Constants

Specify the size (in bytes) of a value encoded in the procedure information for a routine.

```
enum {
    kNoByteCode = 0,
    kOneByteCode = 1,
    kTwoByteCode = 2,
    kFourByteCode = 3
};
```

Constants

`kNoByteCode`

The value occupies no bytes.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kOneByteCode`
The value occupies 1 byte.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kTwoByteCode`
The value occupies 2 bytes.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kFourByteCode`
The value occupies 4 bytes.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

Discussion
These constants are used by the `ProcInfoType` (page 1446) to specify the size (in bytes) of a value encoded in a routine's procedure information.

ProcInfo Field Offset And Width Constants

Specify offsets to fields and the widths of the fields within a value.

```
enum {
    kCallingConventionWidth = 4,
    kCallingConventionPhase = 0,
    kCallingConventionMask = 0x0F,
    kResultSizeWidth = 2,
    kResultSizePhase = kCallingConventionWidth,
    kResultSizeMask = 0x30,
    kStackParameterWidth = 2,
    kStackParameterPhase = (kCallingConventionWidth + kResultSizeWidth),
    kStackParameterMask = 0xFFFFFFC0,
    kRegisterResultLocationWidth = 5,
    kRegisterResultLocationPhase = (kCallingConventionWidth + kResultSizeWidth),
    kRegisterParameterWidth = 5,
    kRegisterParameterPhase = (kCallingConventionWidth + kResultSizeWidth
+ kRegisterResultLocationWidth),
    kRegisterParameterMask = 0x7FFFF800,
    kRegisterParameterSizePhase = 0,
    kRegisterParameterSizeWidth = 2,
    kRegisterParameterWhichPhase = kRegisterParameterSizeWidth,
    kRegisterParameterWhichWidth = 3,
    kDispatchedSelectorSizeWidth = 2,
    kDispatchedSelectorSizePhase = (kCallingConventionWidth + kResultSizeWidth),
    kDispatchedParameterPhase = (kCallingConventionWidth + kResultSizeWidth
+ kDispatchedSelectorSizeWidth),
    kSpecialCaseSelectorWidth = 6,
    kSpecialCaseSelectorPhase = kCallingConventionWidth,
    kSpecialCaseSelectorMask = 0x03F0
};
```

Constants

`kCallingConventionWidth`
The number of bits in the procedure information that encode the calling convention information.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kCallingConventionPhase`
The offset from the least significant bit in the procedure information to the calling convention information.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kCallingConventionMask`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kResultSizeWidth`
The number of bits in the procedure information that encode the function result size information.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kResultSizePhase`
The offset from the least significant bit in the procedure information to the function result size information.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kResultSizeMask`
A mask for the bits in the procedure information that encode the function result size information.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kStackParameterWidth`
The number of bits in the procedure information that encode the size of a stack-based parameter.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kStackParameterPhase`
The offset from the least significant bit in the procedure information to the stack parameter information.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kStackParameterMask`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterResultLocationWidth`
The number of bits in the procedure information that encode which register the result will be stored in.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterResultLocationPhase`
The offset from the least significant bit in the procedure information to the result register information.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterParameterWidth`
The number of bits in the procedure information that encode the information about a register-based parameter.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterParameterPhase`
The offset from the least significant bit in the procedure information to the register parameter information.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterParameterMask`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterParameterSizePhase`
The offset from the beginning of a register parameter information field to the encoded size of the parameter.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterParameterSizeWidth`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterParameterWhichPhase`
The offset from the beginning of a register parameter information field to the encoded register.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterParameterWhichWidth`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kDispatchedSelectorSizeWidth`
The number of bits in the procedure information that encode the size of a routine-dispatching selector.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kDispatchedSelectorSizePhase`
The offset from the least significant bit in the procedure information to the selector size information of a routine that is dispatched though a selector.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kDispatchedParameterPhase`
The offset from the least significant bit in the procedure information to the parameter information of a routine that is dispatched though a selector.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseSelectorWidth`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseSelectorPhase`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseSelectorMask`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion
The offsets to fields and the widths of the fields within a value of type `ProcInfoType` (page 1446) are defined by constants.

Register Constants

Specify registers that are encoded in the procedure information for a routine.

```
enum {
    kRegisterD0 = 0,
    kRegisterD1 = 1,
    kRegisterD2 = 2,
    kRegisterD3 = 3,
    kRegisterD4 = 8,
    kRegisterD5 = 9,
    kRegisterD6 = 10,
    kRegisterD7 = 11,
    kRegisterA0 = 4,
    kRegisterA1 = 5,
    kRegisterA2 = 6,
    kRegisterA3 = 7,
    kRegisterA4 = 12,
    kRegisterA5 = 13,
    kRegisterA6 = 14,
    kCCRegisterCBit = 16,
    kCCRegisterVBit = 17,
    kCCRegisterZBit = 18,
    kCCRegisterNBit = 19,
    kCCRegisterXBit = 20
};
typedef unsigned short registerSelectorType;
```

Constants

`kRegisterD0`
Register D0.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterD1`
Register D1.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterD2`
Register D2.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterD3`
Register D3.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterD4`
Register D4.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterD5`
Register D5.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterD6`
Register D6.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterD7`
Register D7.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterA0`
Register A0.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterA1`
Register A1.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterA2`
Register A2.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterA3`
Register A3.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterA4`
Register A4.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterA5`
Register A5.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterA6`
Register A6.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kCCRegisterCBit`
The C (carry) flag of the Status Register.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kCCRegisterVBit`
The V (overflow) flag of the Status Register.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kCCRegisterZBit`
The Z (zero) flag of the Status Register.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kCCRegisterNBit`
The N (negative) flag of the Status Register.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kCCRegisterXBit`
The X (extend) flag of the Status Register.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion
For register-based routines, the registers are encoded in the routine's procedure information using these constants.

Routine Descriptor Flags

Specify attributes of the described routine.

```
enum {
    kSelectorsAreNotIndexable = 0x00,
    kSelectorsAreIndexable = 0x01
};
```

Constants
`kSelectorsAreNotIndexable`
For dispatched routines, the recognized routine selectors are not contiguous.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSelectorsAreIndexable`
For dispatched routines, the recognized routine selectors are contiguous and therefore indexable.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion
The `routineDescriptorFlags` field of a routine descriptor contains a set of routine descriptor flags that specify attributes of the described routine. You can use constants to specify the routine descriptor flags. In general, you should use the constant `kSelectorsAreNotIndexable` when constructing your own routine descriptors; the value `kSelectorsAreIndexable` is reserved for use by Apple.

Routine Entry Point Flags

Specify information about the entry point for a routine.

```
enum {
    kProcDescriptorIsAbsolute = 0x00,
    kProcDescriptorIsRelative = 0x01
};
```

Constants

`kProcDescriptorIsAbsolute`

The address of the routine's entry point specified in the `procDescriptor` field of a routine record is an absolute address.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kProcDescriptorIsRelative`

The address of the routine's entry point specified in the `procDescriptor` field of a routine record is relative to the beginning of the routine descriptor. If the code is contained in a resource and its absolute location is not known until run time, you should set this flag.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion

The `routineFlags` field of a routine record contains a set of flags that specify information about a routine. You can use constants to specify the desired routine flags. Currently, only 5 of the 16 bits in a routine flags word are defined. You should set all the other bits to 0.

Routine Selector Flags

Specify whether or not to pass a selector to a routine.

```
enum {
    kPassSelector = 0x00,
    kDontPassSelector = 0x08
};
```

Constants

`kPassSelector`

Pass the routine selector to the target routine as a parameter.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kDontPassSelector`

Do not pass the routine selector to the target routine as a parameter. You should not use this flag for 680x0 routines.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion

The `routineFlags` field of a routine record contains a set of flags that specify information about a routine. You can use constants to specify the desired routine flags. Currently, only 5 of the 16 bits in a routine flags word are defined. You should set all the other bits to 0.

In general, you should use the constant `kPassSelector`. The constant `kDontPassSelector` is reserved for use with selector-based system software routines.

Special Case Calling Convention Constants

Specify the calling conventions for a routine.

```
enum {
    kSpecialCaseHighHook = 0,
    kSpecialCaseCaretHook = 0,
    kSpecialCaseEOLHook = 1,
    kSpecialCaseWidthHook = 2,
    kSpecialCaseTextWidthHook = 2,
    kSpecialCaseNWidthHook = 3,
    kSpecialCaseDrawHook = 4,
    kSpecialCaseHitTestHook = 5,
    kSpecialCaseTEFindWord = 6,
    kSpecialCaseProtocolHandler = 7,
    kSpecialCaseSocketListener = 8,
    kSpecialCaseTERecalc = 9,
    kSpecialCaseTEDoText = 10,
    kSpecialCaseGNEFilterProc = 11,
    kSpecialCaseMBarHook = 12
};
```

Constants

`kSpecialCaseHighHook`

The routine follows the calling conventions documented in Inside Macintosh: Text; a rectangle is on the stack and a pointer is in register A3; no result is returned.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseCaretHook`

The routine follows the calling conventions documented in Inside Macintosh: Text; a rectangle is on the stack and a pointer is in register A3; no result is returned.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseEOLHook`

Parameters are passed to the routine in registers A3, A4, and D0, and output is returned in the Z flag of the Status Register. An `EOLHook` routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseWidthHook`

Parameters are passed to the routine in registers A0, A3, A4, D0, and D1, and output is returned in register D1. A `WIDTHHook` routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseTextWidthHook`
Parameters are passed to the routine in registers A0, A3, A4, D0, and D1, and output is returned in register D1. A `TextWidthHook` routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseNWidthHook`
Parameters are passed to the routine in registers A0, A2, A3, A4, D0, and D1, and output is returned in register D1. An `nWIDTHHook` routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseDrawHook`
Parameters are passed to the routine in registers A0, A3, A4, D0, and D1, and no result is returned. A `DRAWHook` routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseHitTestHook`
Parameters are passed to the routine in registers A0, A3, A4, D0, D1, and D2, and output is returned in registers D0, D1, and D2. A `HITTESTHook` routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseTEFindWord`
Parameters are passed to the routine in registers A3, A4, D0, and D2, and output is returned in registers D0 and D1. A `TEFindWord` hook has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseProtocolHandler`
Parameters are passed to the routine in registers A0, A1, A2, A3, A4, and in the low-order word of register D1; output is returned in the Z flag of the Status Register. A protocol handler has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseSocketListener`
Parameters are passed to the routine in registers A0, A1, A2, A3, A4, in the low-order byte of register D0, and in the low-order word of register D1; output is returned in the Z flag of the Status Register. A socket listener has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseTERecalc`
Parameters are passed to the routine in registers A3 and D7, and output is returned in registers D2, D3, and D4. A TextEdit line-start recalculation routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseTEDoText`
Parameters are passed to the routine in registers A3, D3, D4, and D7, and output is returned in registers A0 and D0. A TextEdit text-display, hit-test, and caret-positioning routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseGNEFilterProc`
Parameters are passed to the routine in registers A1 and D0 and on the stack, and output is returned on the stack. A `GetNextEvent` filter procedure has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseMBarHook`
Parameters are passed to the routine on the stack, and output is returned in register D0. A menu bar hook routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion
These constants are used by the `ProcInfoType` (page 1446) type to specify a routine's calling conventions.

Result Codes

The most common result codes returned by the Mixed Mode Manager are listed below.

Result Code	Value	Description
`mmInternalError`	-2526	An internal error has occurred. Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availabilty information by using the Mixed Mode Manager selectors defined in the Gestalt Manager. For more information see *Inside Mac OS X: Gestalt Manager Reference*.

Multiprocessing Services Reference

Framework:	CoreServices/CoreServices.h
Declared in	Multiprocessing.h MultiprocessingInfo.h
Companion guide	Multiprocessing Services Programming Guide

Overview

Multiprocessing Services is an API that lets you create preemptive tasks in your application that can run on one or more microprocessors. Unlike the cooperative threads created by the Thread Manager, Multiprocessing Services automatically divides processor time among the available tasks, so that no particular task can monopolize the system. This document is relevant to you if you want to add multitasking capability to your Mac OS applications.

In Mac OS X, Carbon supports Multiprocessing Services with the following restrictions:

- Debugging functions are not implemented. Use the mach APIs provided by the system to implement debugging services.
- Opaque notification IDs are local to your process; they are not globally addressable across processes.
- Global memory allocation is not supported.

Functions by Task

Determining Multiprocessing Services And Processor Availability

`_MPIsFullyInitialized` (page 1508)
Indicates whether Multiprocessing Services is available for use.

`MPGetNextCpuID` (page 1489)
Obtains the next CPU ID in the list of physical processors of the specified memory coherence group.

`MPProcessors` (page 1493)
Returns the number of processors on the host computer.

`MPProcessorsScheduled` (page 1493)
Returns the number of active processors available on the host computer.

Creating and Handling Message Queues

`MPCreateQueue` (page 1478)
Creates a message queue.

`MPDeleteQueue` (page 1484)
Deletes a message queue.

`MPNotifyQueue` (page 1492)
Sends a message to the specified message queue.

`MPSetQueueReserve` (page 1497)
Reserves space for messages on a specified message queue.

`MPWaitOnQueue` (page 1506)
Obtains a message from a specified message queue.

Creating and Handling Semaphores

`MPCreateSemaphore` (page 1478)
Creates a semaphore.

`MPDeleteSemaphore` (page 1485)
Removes a semaphore.

`MPSignalSemaphore` (page 1502)
Signals a semaphore.

`MPWaitOnSemaphore` (page 1507)
Waits on a semaphore

Creating and Scheduling Tasks

`MPCreateTask` (page 1479)
Creates a preemptive task.

`MPCurrentTaskID` (page 1481)
Obtains the task ID of the currently-executing preemptive task

`MPSetTaskType` (page 1499)
Sets the type of the task.

`MPExit` (page 1487)
Allows a task to terminate itself

`MPGetNextTaskID` (page 1490)
Obtains the next task ID in the list of available tasks.

`MPSetTaskWeight` (page 1500)
Assigns a relative weight to a task, indicating how much processor time it should receive compared to other available tasks.

`MPTaskIsPreemptive` (page 1502)
Determines whether a task is preemptively scheduled.

`MPTerminateTask` (page 1503)
Terminates an existing task.

MPYield (page 1507)
Allows a task to yield the processor to another task.

Handling Critical Regions

MPCreateCriticalRegion (page 1476)
Creates a critical region object.

MPDeleteCriticalRegion (page 1483)
Removes the specified critical region object.

MPEnterCriticalRegion (page 1486)
Attempts to enter a critical region.

MPExitCriticalRegion (page 1487)
Exits a critical region.

Handling Event Groups

MPCreateEvent (page 1477)
Creates an event group.

MPDeleteEvent (page 1483)
Removes an event group.

MPSetEvent (page 1496)
Merges event flags into a specified event group.

MPWaitForEvent (page 1505)
Retrieves event flags from a specified event group.

Handling Kernel Notifications

MPCauseNotification (page 1476)
Signals a kernel notification.

MPCreateNotification (page 1477)
Creates a kernel notification

MPDeleteNotification (page 1484)
Removes a kernel notification.

MPModifyNotification (page 1491)
Adds a simple notification to a kernel notification.

MPModifyNotificationParameters (page 1492)

Accessing Per-Task Storage Variables

MPAllocateTaskStorageIndex (page 1472)
Returns an index number to access per-task storage.

`MPDeallocateTaskStorageIndex` (page 1482)
Frees an index number used to access per-task storage

`MPGetTaskStorageValue` (page 1490)
Gets the storage value stored at a specified index number.

`MPSetTaskStorageValue` (page 1499)
Sets the storage value for a given index number.

Memory Allocation Functions

`MPAllocate` (page 1471)
Allocates a nonrelocatable memory block. (Deprecated. Use `MPAllocateAligned` instead.)

`MPAllocateAligned` (page 1472)
Allocates a nonrelocatable memory block.

`MPBlockClear` (page 1474)
Clears a block of memory.

`MPBlockCopy` (page 1474)
Copies a block of memory.

`MPDataToCode` (page 1481)
Designates the specified block of memory as executable code.

`MPFree` (page 1488)
Frees memory allocated by `MPAllocateAligned`.

`MPGetAllocatedBlockSize` (page 1489)
Returns the size of a memory block.

Remote Calling Functions

`MPRemoteCall` (page 1494)
Calls a non-reentrant function and blocks the current task.

`MPRemoteCallCFM` (page 1495)
Calls a non-reentrant function and blocks the current task.

Timer Services Functions

`MPArmTimer` (page 1473)
Arms the timer to expire at a given time.

`MPCancelTimer` (page 1475)
Cancels an armed timer.

`MPCreateTimer` (page 1480)
Creates a timer.

`MPDelayUntil` (page 1482)
Blocks the calling task until a specified time.

`MPDeleteTimer` (page 1485)
Removes a timer.

`MPSetTimerNotify` (page 1500)
Sets the notification information associated with a timer.

Exception Handling Functions

`MPDisposeTaskException` (page 1486)
Removes a task exception.

`MPExtractTaskState` (page 1488)
Extracts state information from a suspended task.

`MPSetExceptionHandler` (page 1496)
Sets an exception handler for a task.

`MPSetTaskState` (page 1498)
Sets state information for a suspended task.

`MPThrowException` (page 1504)
Throws an exception to a specified task.

Debugger Support Functions

`MPRegisterDebugger` (page 1494)
Registers a debugger.

`MPUnregisterDebugger` (page 1504)
Unregisters a debugger.

Functions

MPAllocate

Allocates a nonrelocatable memory block. (**Deprecated.** Use `MPAllocateAligned` instead.)

```
LogicalAddress MPAllocate (
    ByteCount size
);
```

Parameters

size
The size, in bytes, of the memory block to allocate.

Return Value
A pointer to the allocated memory. If the function cannot allocate the requested memory or the requested alignment, the returned address is `NULL`.

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

Declared In
`Multiprocessing.h`

MPAllocateAligned

Allocates a nonrelocatable memory block.

```
LogicalAddress MPAllocateAligned (
   ByteCount size,
   UInt8 alignment,
   OptionBits options
);
```

Parameters

size

The size, in bytes, of the memory block to allocate.

alignment

The desired alignment of the allocated memory block. See "Memory Allocation Alignment Constants" (page 1523) for a list of possible values to pass. Note that there will be a minimum alignment regardless of the requested alignment. If the requested memory block is 4 bytes or smaller, the block will be at least 4-byte aligned. If the requested block is greater than 4 bytes, the block will be at least 8-byte aligned.

options

Any optional information to use with this call. See "Memory Allocation Option Constants" (page 1525) for a list of possible values to pass.

Return Value

A pointer to the allocated memory. If the function cannot allocate the requested memory or the requested alignment, the returned address is `NULL`.

Discussion

The memory referenced by the returned address is guaranteed to be accessible by the application's cooperative task and any preemptive tasks that it creates, but not by other applications or their preemptive tasks. Any existing non-global heap blocks are freed when the application terminates. As with all shared memory, you must explicitly synchronize access to allocated heap blocks using a notification mechanism.

You can replicate the effect of the older `MPAllocate` function by calling `MPAllocateAligned` with 32-byte alignment and no options.

Also see the function `MPFree` (page 1488).

Special Considerations

Mac OS X does not support allocation of global (cross address space) or resident memory with this function. In addition, passing the `kMPAllocateNoGrowthMask` constant in the `options` parameter has no effect in Mac OS X, since memory allocation is done with sparse heaps.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Multiprocessing.h`

MPAllocateTaskStorageIndex

Returns an index number to access per-task storage.

```
OSStatus MPAllocateTaskStorageIndex (
   TaskStorageIndex *taskIndex
);
```

Parameters

index

On return, `index` contains an index number you can use to store task data.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
A call to the function `MPAllocateTaskStorageIndex` returns an index number that is common across all tasks in the current process. You can use this index number in calls to `MPSetTaskStorageValue` (page 1499) and `MPGetTaskStorageValue` (page 1490) to set a different value for each task using the same index.

You can think of the task storage area as a two dimensional array cross-referenced by the task storage index number and the task ID. Note that since the amount of per-task storage is determined when the task is created, the number of possible index values associated with a task is limited.

Also see the function `MPDeallocateTaskStorageIndex` (page 1482).

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

Declared In
`Multiprocessing.h`

MPArmTimer

Arms the timer to expire at a given time.

```
OSStatus MPArmTimer (
   MPTimerID timerID,
   AbsoluteTime *expirationTime,
   OptionBits options
);
```

Parameters

timerID

The ID of the timer you want to arm.

expirationTime

A pointer to a value that specifies when you want the timer to expire. Note that if you arm the timer with a time that has already passed, the timer expires immediately.

options

Any optional actions. See "Timer Option Masks" (page 1532) for a list of possible values.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533). If the timer has already expired, the reset does not take place and the function returns `kMPInsufficientResourcesErr`.

Discussion
The expiration time is an absolute time, which you can generate by calling the Driver Services Library function `UpTime`. When the timer expires, a notification is sent to the notification mechanism specified in the last `MPSetTimerNotify` (page 1500) call. If the specified notification ID has become invalid, no action is taken when the timer expires. The timer itself is deleted when it expires unless you specified the `kMPPreserveTimerID` option in the options parameter.

Also see the function `MPCancelTimer` (page 1475).

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

Declared In
`Multiprocessing.h`

MPBlockClear

Clears a block of memory.

```
void MPBlockClear (
   LogicalAddress address,
   ByteCount size
);
```

Parameters

address
The starting address of the memory block you want to clear.

size
The number of bytes you want to clear.

Discussion
As with all shared memory, your application must synchronize access to the memory blocks to avoid data corruption. `MPBlockClear` ensures the clearing stays within the bounds of the area specified by `size`, but the calling task can be preempted during the copying process.

Note that you can call this function from an interrupt handler.

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

Declared In
`Multiprocessing.h`

MPBlockCopy

Copies a block of memory.

```
void MPBlockCopy (
   LogicalAddress source,
   LogicalAddress destination,
   ByteCount size
);
```

Parameters

source

The starting address of the memory block you want to copy.

destination

The location to which you want to copy the memory block.

size

The number of bytes to copy.

Discussion

This function simply calls through to the Driver Services Library function `BlockMoveData`. Note that you should not make any assumptions about the state of the destination memory while this function is executing. In the intermediate state, values may be present that are neither the original nor the final ones. For example, this function may use the `'dcbz'` instruction. If the underlying memory is not cacheable, if the memory is write-through instead of copy-back, or if the cache block is flushed for some reason, the `'dcbz'` instruction will write zeros to the destination. You can avoid the use of the `'dcbz'` instruction by calling `BlockMoveDataUncached`, but even that function makes no other guarantees about the memory block's intermediate state.

As with all shared memory, your application must synchronize access to the memory blocks to avoid data corruption. `MPBlockCopy` ensures the copying stays within the bounds of the area specified by `size`, but the calling task can be preempted during the copying process.

Note that you can call this function from an interrupt handler.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Multiprocessing.h`

MPCancelTimer

Cancels an armed timer.

```
OSStatus MPCancelTimer (
   MPTimerID timerID,
   AbsoluteTime *timeRemaining
);
```

Parameters

timerID

The ID of the armed timer you want to cancel.

timeRemaining

On return, the `timeRemaining` contains the time remaining before the timer would have expired.

Return Value

A result code. See "Multiprocessing Services Result Codes" (page 1533). If the timer has already expired, this function returns `kMPInsufficientResourcesErr`.

Discussion
Also see the function `MPArmTimer` (page 1473).

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

Declared In
`Multiprocessing.h`

MPCauseNotification

Signals a kernel notification.

```
OSStatus MPCauseNotification (
   MPNotificationID notificationID
);
```

Parameters

notificationID
The ID of the kernel notification you want to signal.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
You call this function to signal a kernel notification much as you would signal any simple notification (for example, `MPNotifyQueue` (page 1492)).

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

Declared In
`Multiprocessing.h`

MPCreateCriticalRegion

Creates a critical region object.

```
OSStatus MPCreateCriticalRegion (
   MPCriticalRegionID *criticalRegion
);
```

Parameters

criticalRegion
On return, the `criticalRegion` contains the ID of the newly created critical region object.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
Also see the function `MPDeleteCriticalRegion` (page 1483).

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

Declared In
`Multiprocessing.h`

MPCreateEvent

Creates an event group.

```
OSStatus MPCreateEvent (
   MPEventID *event
);
```

Parameters

event

On return, `event` contains the ID of the newly created event group.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
Event groups are created from dynamically allocated internal resources. Other tasks may be competing for these resources so it is possible that this function will not be able to create an event group.

Also see the function `MPDeleteEvent` (page 1483).

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

Declared In
`Multiprocessing.h`

MPCreateNotification

Creates a kernel notification

```
OSStatus MPCreateNotification (
   MPNotificationID *notificationID
);
```

Parameters

notificationID

On return, `notificationID` points to the newly created kernel notification.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
After creating the kernel notification object, you can add simple notifications by calling the function `MPModifyNotification` (page 1491).

Also see the function `MPDeleteNotification` (page 1484).

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

Declared In
`Multiprocessing.h`

MPCreateQueue

Creates a message queue.

```
OSStatus MPCreateQueue (
   MPQueueID *queue
);
```

Parameters

queue
On return, the variable contains the ID of the newly created message queue.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533). If a queue could not be created, `MPCreateQueue` returns `kMPInsufficientResourcesErr`.

Discussion
This call creates a message queue, which can be used to notify (that is, send) and wait for (that is, receive) messages consisting of three pointer-sized values in a preemptively safe manner.

Message queues are created from dynamically allocated internal resources. Other tasks may be competing for these resources so it is possible this function may not be able to create a queue.

See also the functions `MPDeleteQueue` (page 1484) and `MPSetQueueReserve` (page 1497).

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

Declared In
`Multiprocessing.h`

MPCreateSemaphore

Creates a semaphore.

```
OSStatus MPCreateSemaphore (
   MPSemaphoreCount maximumValue,
   MPSemaphoreCount initialValue,
   MPSemaphoreID *semaphore
);
```

Parameters

maximumValue
The maximum allowed value of the semaphore.

initialValue
The initial value of the semaphore.

semaphore
On return, `semaphore` contains the ID of the newly–created semaphore.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
If you want to create a binary semaphore, you can call the macro `MPCreateBinarySemaphore (MPSemaphoreID *semaphore)` instead, which simply calls `MPCreateSemaphore` with both `maximumValue` and `initialValue` set to 1.

Also see the function `MPDeleteSemaphore` (page 1485).

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

Declared In
`Multiprocessing.h`

MPCreateTask

Creates a preemptive task.

```
OSStatus MPCreateTask (
   TaskProc entryPoint,
   void *parameter,
   ByteCount stackSize,
   MPQueueID notifyQueue,
   void *terminationParameter1,
   void *terminationParameter2,
   MPTaskOptions options,
   MPTaskID *task
);
```

Parameters

entryPoint
A pointer to the task function. The task function should take a single pointer-sized parameter and return a value of type `OSStatus`.

parameter
The parameter to pass to the task function.

stackSize
The size of the stack assigned to the task. Note that you should be careful not to exceed the bounds of the stack, since stack overflows may not be detected. Specifying zero for the size will result in a default stack size of 4KB.

Note that in Mac OS X prior to version 10.1, this parameter is ignored, and all stacks have the default size of 512 KB. Versions 10.1 and later do not have this limitation.

notifyQueue
The ID of the message queue to which the system will send a message when the task terminates. You specify the first two values of the message in the parameters `terminationParameter1` and `terminationParameter2` respectively. The last message value contains the result code of the task function.

terminationParameter1
A pointer-sized value that is sent to the message queue specified by the parameter `notifyQueue` when the task terminates.

terminationParameter2
A pointer-sized value that is sent to the message queue specified by the parameter `notifyQueue` when the task terminates.

options

Optional attributes of the preemptive task. See "Task Creation Options" (page 1528) for a list of possible values.

task

On return, `task` points to the ID of the newly created task.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533). If `MPCreateTask` could not create the task because some critical resource was not available, the function returns `kMPInsufficientResourcesErr`. Usually this is due to lack of memory to allocate the internal data structures associated with the task or the stack. The function also returns `kMPInsufficientResourcesErr` if any reserved option bits are set.

Discussion
Tasks are created in the unblocked state, ready for execution. A task can terminate in the following ways:

- By returning from its entry point
- By calling `MPExit` (page 1487)
- When specified as the target of an `MPTerminateTask` (page 1503) call
- If a hardware-detected exception or programming exception occurs and no exception handler is installed
- If the application calls `ExitToShell`

Task resources (its stack, active timers, internal structures related to the task, and so on) are reclaimed by the system when the task terminates. The task's address space is inherited from the process address space. All existing tasks are terminated when the owning process terminates.

To set the relative processor weight to be assigned to a task, use the function `MPSetTaskWeight` (page 1500).

See also the function `MPTerminateTask` (page 1503).

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

Declared In
`Multiprocessing.h`

MPCreateTimer

Creates a timer.

```
OSStatus MPCreateTimer (
   MPTimerID *timerID
);
```

Parameters

timerID

On return, the `timerID` contains the ID of the newly created timer.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
You can use a timer to notify an event, queue, or semaphore after a specified amount of time has elapsed.

Timer objects are created from dynamically-allocated internal resources. Other tasks may be competing for these resources so it is possible this function may not be able to create one.

To specify the notification mechanism to signal, use the function `MPSetTimerNotify` (page 1500).

Also see the functions `MPDeleteTimer` (page 1485) and `MPArmTimer` (page 1473).

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

Declared In
`Multiprocessing.h`

MPCurrentTaskID

Obtains the task ID of the currently-executing preemptive task

```
MPTaskID MPCurrentTaskID (
   void
);
```

Return Value
The task ID of the current preemptive task. See the description of the `MPTaskID` data type.

Discussion
Returns the ID of the current preemptive task. If called from a cooperative task, this function returns an ID which is different than the ID of any preemptive task. Nonpreemptive processes may or may not have different task IDs for each application; future implementations of this API may behave differently in this regard.

Note that you can call this function from an interrupt handler.

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

Declared In
`Multiprocessing.h`

MPDataToCode

Designates the specified block of memory as executable code.

```
void MPDataToCode (
   LogicalAddress address,
   ByteCount size
);
```

Parameters

address
The starting address of the memory block you want to designate as code.

size
The size of the memory block.

Discussion
Since processors need to differentiate between code and data in memory, you should call this function to tag any executable code that your tasks may generate.

Note that you can call this function from an interrupt handler.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Multiprocessing.h`

MPDeallocateTaskStorageIndex

Frees an index number used to access per-task storage

```
OSStatus MPDeallocateTaskStorageIndex (
   TaskStorageIndex taskIndex
);
```

Parameters
index
The index number you want to deallocate.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
Also see the function `MPAllocateTaskStorageIndex` (page 1472).

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

Declared In
`Multiprocessing.h`

MPDelayUntil

Blocks the calling task until a specified time.

```
OSStatus MPDelayUntil (
   AbsoluteTime *expirationTime
);
```

Parameters
expirationTime
The time to unblock the task.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
You cannot call this function from a cooperative task.

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

Declared In
`Multiprocessing.h`

MPDeleteCriticalRegion

Removes the specified critical region object.

```
OSStatus MPDeleteCriticalRegion (
   MPCriticalRegionID criticalRegion
);
```

Parameters

criticalRegion
The critical region object you want to remove.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
Calling this function unblocks all tasks waiting to enter the critical region and their respective `MPEnterCriticalRegion` (page 1486) calls will return with the result code `kMPDeletedErr`.

Also see the function `MPCreateCriticalRegion` (page 1476).

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

Declared In
`Multiprocessing.h`

MPDeleteEvent

Removes an event group.

```
OSStatus MPDeleteEvent (
   MPEventID event
);
```

Parameters

event
The ID of the event group you want to remove.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
After deletion, the event ID becomes invalid, and all internal resources associated with the event group are reclaimed. Calling this function unblocks all tasks waiting on the event group and their respective `MPWaitForEvent` (page 1505) calls will return with the result code `kMPDeletedErr`.

Also see the function `MPCreateEvent` (page 1477).

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

Declared In
`Multiprocessing.h`

MPDeleteNotification

Removes a kernel notification.

```
OSStatus MPDeleteNotification (
   MPNotificationID notificationID
);
```

Parameters

notificationID
The ID of the notification you want to remove.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
Also see the function `MPCreateNotification` (page 1477).

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

Declared In
`Multiprocessing.h`

MPDeleteQueue

Deletes a message queue.

```
OSStatus MPDeleteQueue (
   MPQueueID queue
);
```

Parameters

queue
The ID of the message queue you want to delete.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
After calling `MPDeleteQueue`, the specified queue ID becomes invalid, and all internal resources associated with the queue (including queued messages) are reclaimed. Any tasks waiting on the queue are unblocked and their respective `MPWaitOnQueue` (page 1506) calls will return with the result code `kMPDeletedErr`.

Also see the function `MPCreateQueue` (page 1478).

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

Declared In
`Multiprocessing.h`

MPDeleteSemaphore

Removes a semaphore.

```
OSStatus MPDeleteSemaphore (
   MPSemaphoreID semaphore
);
```

Parameters

semaphore
The ID of the semaphore you want to remove.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
Calling this function unblocks all tasks waiting on the semaphore and the tasks' respective `MPWaitOnSemaphore` (page 1507) calls will return with the result code `kMPDeletedErr`.

Also see the function `MPCreateSemaphore` (page 1478).

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

Declared In
`Multiprocessing.h`

MPDeleteTimer

Removes a timer.

```
OSStatus MPDeleteTimer (
   MPTimerID timerID
);
```

Parameters

timerID
The ID of the timer you want to remove.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
After deletion, the timer ID becomes invalid, and all internal resources associated with the timer are reclaimed.

Also see the function `MPCreateTimer` (page 1480).

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

Declared In
`Multiprocessing.h`

MPDisposeTaskException

Removes a task exception.

```
OSStatus MPDisposeTaskException (
   MPTaskID task,
   OptionBits action
);
```

Parameters

task

The task whose exception you want to remove.

action

Any actions to perform on the task. For example, you can enable single-stepping when the task resumes, or you can pass the exception on to another handler. See "Task Exception Disposal Constants" (page 1528) for a listing of possible values.

Return Value

A result code. See "Multiprocessing Services Result Codes" (page 1533). If the specified action is invalid or unsupported, or if the specified task is not suspended, this function returns `kMPInsufficientResourcesErr`.

Discussion

This function removes the task exception and allows the task to resume operation. If desired, you can enable single-stepping or branch-stepping, or propagate the exception instead.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Multiprocessing.h`

MPEnterCriticalRegion

Attempts to enter a critical region.

```
OSStatus MPEnterCriticalRegion (
   MPCriticalRegionID criticalRegion,
   Duration timeout
);
```

Parameters

criticalRegion

The ID of the critical region you want to enter.

timeout

The maximum time to wait for entry before timing out. See "Timer Duration Constants" (page 1531) for a list of constants you can use to specify the wait interval.

Return Value

A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion

If another task currently occupies the critical region, the current task is blocked until the critical region is released or until the designated timeout expires. Otherwise the task enters the critical region and `MPEnterCriticalRegion` increments the region's use count.

Once a task enters a critical region it can make further calls to `MPEnterCriticalRegion` without blocking (its use count increments for each call). However, each call to `MPEnterCriticalRegion` must be balanced by a call to `MPExitCriticalRegion` (page 1487) ; otherwise the region is not released for use by other tasks.

Note that you can enter a critical region from a cooperative task. Each cooperative task is treated as unique and different from any preemptive task. If you call this function from a cooperative task, you should specify only `kDurationImmediate` for the timeout length; other waits will cause the task to block.

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

Declared In
`Multiprocessing.h`

MPExit

Allows a task to terminate itself

```
void MPExit (
   OSStatus status
);
```

Parameters

`status`

An application-defined value that indicates termination status. This value is sent to the termination message queue in place of the task's result code.

Discussion
When called from within a preemptive task, the task terminates, and the value indicated by the parameter `status` is sent to the termination message queue you specified in `MPCreateTask` (page 1479). Note that you cannot call `MPExit` from outside a preemptive task.

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

Declared In
`Multiprocessing.h`

MPExitCriticalRegion

Exits a critical region.

```
OSStatus MPExitCriticalRegion (
   MPCriticalRegionID criticalRegion
);
```

Parameters

`criticalRegion`

The ID of the critical region you want to exit.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533). If the task does not own the critical region specified by `criticalRegion`, `MPExitCriticalRegion` returns `kMPInsufficientResourcesErr`.

Discussion
This function decrements the use count of the critical region object. When the use count reaches zero, ownership of the critical region object is released (which allows another task to use the critical region).

Also see the function `MPEnterCriticalRegion` (page 1486).

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

Declared In
`Multiprocessing.h`

MPExtractTaskState

Extracts state information from a suspended task.

```
OSStatus MPExtractTaskState (
   MPTaskID task,
   MPTaskStateKind kind,
   void *info
);
```

Parameters

task

The task whose state information you want to obtain.

kind

The kind of state information you want to obtain. See "Task State Constants" (page 1530) for a listing of possible values.

info

A pointer to a data structure to hold the state information. On return, the data structure holds the desired state information. The format of the data structure varies depending on the state information you want to retrieve. See the header file `MachineExceptions.h` for the formats of the various state information structures.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533). If you attempt to extract state information for a running task, this function returns `kMPInsufficientResourcesErr`.

Discussion
You can use this function to obtain register contents or exception information about a particular task.

Also see the function `MPSetTaskState` (page 1498).

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

Declared In
`Multiprocessing.h`

MPFree

Frees memory allocated by `MPAllocateAligned`.

```
void MPFree (
   LogicalAddress object
);
```

Parameters

object

A pointer to the memory you want to release.

Discussion
Also see the function `MPAllocateAligned` (page 1472).

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

Declared In
`Multiprocessing.h`

MPGetAllocatedBlockSize

Returns the size of a memory block.

```
ByteCount MPGetAllocatedBlockSize (
   LogicalAddress object
);
```

Parameters

object

The address of the memory block whose size you want to determine.

Return Value
The size of the allocated memory block, in bytes.

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

Declared In
`Multiprocessing.h`

MPGetNextCpuID

Obtains the next CPU ID in the list of physical processors of the specified memory coherence group.

```
OSStatus MPGetNextCpuID (
   MPCoherenceID owningCoherenceID,
   MPCpuID *cpuID
);
```

Parameters

owningCoherenceID

The ID of the memory coherence group whose physical processor IDs you want to obtain. Pass `kMPInvalidIDErr`, as only one coherence group, internal RAM, is currently defined.

cpuID

On return, `cpuID` points to the ID of the next physical processor.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
By iterating on this function (after calling `MPProcessors` (page 1493) , for example), you can obtain the IDs of all the processors available on the host computer. Generally, you would only use this function in diagnostic programs.

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

Declared In
`MultiprocessingInfo.h`

MPGetNextTaskID

Obtains the next task ID in the list of available tasks.

```
OSStatus MPGetNextTaskID (
   MPProcessID owningProcessID,
   MPTaskID *taskID
);
```

Parameters

owningProcessID
The ID of the process (typically the application) that owns the tasks. This ID is the same as the process ID handled by the Code Fragment Manager.

taskID
On return, `taskID` points to ID of the next task in the list of tasks.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
By iterating on this function, you can obtain the IDs of all the tasks in a given process. These tasks may be running, ready, or blocked. Generally you would only use this function in diagnostic programs.

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

Declared In
`MultiprocessingInfo.h`

MPGetTaskStorageValue

Gets the storage value stored at a specified index number.

```
TaskStorageValue MPGetTaskStorageValue (
   TaskStorageIndex taskIndex
);
```

Parameters

index
The index number of the storage value you want to obtain.

Return Value
The value stored at the specified index number. See the description of the `TaskStorageValue` data type.

Discussion
Calling this function from within a task effectively reads a value in a two-dimensional array cross-referenced by task storage index value and the task ID.

Note that since this function does not return any status information, it may not be immediately obvious whether the returned storage value is valid.

Also see the function `MPSetTaskStorageValue` (page 1499).

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

Declared In
`Multiprocessing.h`

MPModifyNotification

Adds a simple notification to a kernel notification.

```
OSStatus MPModifyNotification (
   MPNotificationID notificationID,
   MPOpaqueID anID,
   void *notifyParam1,
   void *notifyParam2,
   void *notifyParam3
);
```

Parameters

notificationID
The ID of the kernel notification you want to add to..

anID
The ID of the simple notification (semaphore, message group, or event group) you want to add to the kernel notification.

notifyParam1
If `anID` specifies an event group, this parameter should contain the flags to set in the event group when `MPCauseNotification` (page 1476) is called. If `anID` specifies a message queue, this parameter should contain the first pointer-sized value of the message to be sent to the message queue when `MPCauseNotification` (page 1476) is called.

notifyParam2
If `anID` specifies a message queue, this parameter should contain the second pointer-sized value of the message to be sent to the message queue when `MPCauseNotification` (page 1476) is called. Pass `NULL` if you don't need this parameter.

notifyParam3
If `anID` specifies a message queue, this parameter should contain the third pointer-sized value of the message sent to the message queue when `MPCauseNotification` (page 1476) is called. Pass `NULL` if you don't need this parameter.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
You specify the parameters for the simple notifications just as if you were calling the `MPSetTimerNotify` (page 1500) function.

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

Declared In
`Multiprocessing.h`

MPModifyNotificationParameters

```
OSStatus MPModifyNotificationParameters (
   MPNotificationID notificationID,
   MPOpaqueIDClass kind,
   void *notifyParam1,
   void *notifyParam2,
   void *notifyParam3
);
```

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

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

Declared In
`Multiprocessing.h`

MPNotifyQueue

Sends a message to the specified message queue.

```
OSStatus MPNotifyQueue (
   MPQueueID queue,
   void *param1,
   void *param2,
   void *param3
);
```

Parameters

queue
The queue ID of the message queue you want to notify.

param1
The first pointer-sized value of the message to send.

param2
The second pointer-sized value of the message to send.

param3
The third pointer-sized value of the message to send.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
This function sends a message to the specified queue, which consist of the three parameters, `param1`, `param2`, and `param3`. The system does not interpret the three values which comprise the text of the message. If tasks are waiting on the specified queue, the first waiting task is unblocked and the task's `MPWaitOnQueue` (page 1506) function completes.

Depending on the queue mode, the system either allocates messages dynamically or assigns them to memory reserved for the queue. In either case, if no more memory is available for messages `MPNotifyQueue` returns `kMPInsufficientResourcesErr`.

You can call this function from an interrupt handler if messages are reserved on the queue. For more information about queueing modes and reserving messages, see `MPSetQueueReserve` (page 1497).

Also see the function `MPWaitOnQueue` (page 1506).

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

Declared In
`Multiprocessing.h`

MPProcessors

Returns the number of processors on the host computer.

```
ItemCount MPProcessors (
   void
);
```

Return Value
The number of physical processors on the host computer.

Discussion
See also the function `MPProcessorsScheduled` (page 1493).

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

Declared In
`Multiprocessing.h`

MPProcessorsScheduled

Returns the number of active processors available on the host computer.

```
ItemCount MPProcessorsScheduled (
   void
);
```

Return Value
The number of active processors available on the host computer.

Discussion
The number of active processors is defined as the number of processors scheduled to run tasks. This number varies while the system is running. Advanced power management facilities may stop or start scheduling processors in the system to control power consumption or to maintain a proper operating temperature.

See also the function `MPProcessors` (page 1493).

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

Declared In
`Multiprocessing.h`

MPRegisterDebugger

Registers a debugger.

```
OSStatus MPRegisterDebugger (
   MPQueueID queue,
   MPDebuggerLevel level
);
```

Parameters

queue

The ID of the queue to which you want exception messages and other information to be sent.

level

The level of this debugger with respect to other debuggers. Exceptions and informational messages are sent first to the debugger with the highest level. If more than one debugger attempts to register at a particular level, only the first debugger is registered. Other attempts return an error.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
In Mac OS X, this function is available but is not implemented. Use system debugging services to write a debugger for Mac OS X.

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

Declared In
`Multiprocessing.h`

MPRemoteCall

Calls a non-reentrant function and blocks the current task.

```
void * MPRemoteCall (
    MPRemoteProcedure remoteProc,
    void *parameter,
    MPRemoteContext context
);
```

Parameters

remoteProc

A pointer to the application-defined function you want to call. See `MPRemoteProcedure` (page 1508) for more information about the form of this function.

parameter

A pointer to a parameter to pass to the application-defined function. For example, this value could point to a data structure or a memory location.

context

This parameter is ignored; specify `kMPOwningProcessRemoteContext`.

Return Value

The value that your remote procedure callback returned.

Discussion

You use this function to execute code on your application's main task. The *remoteProc* function is scheduled on the application's main run loop and run in the default mode (`kCFRunloopDefaultMode`). If you call this function from your application's main task, the *remoteProc* function is executed immediately in the current mode without blocking the task; otherwise, calling this function blocks the current task until the remote call completes.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Multiprocessing.h`

MPRemoteCallCFM

Calls a non-reentrant function and blocks the current task.

```
void * MPRemoteCallCFM (
   MPRemoteProcedure remoteProc,
   void *parameter,
   MPRemoteContext context
);
```

Parameters

remoteProc

A pointer to the application-defined CFM (Code Fragment Manager) function you want to call. See `MPRemoteProcedure` (page 1508) for more information about the form of this function.

parameter

A pointer to a parameter to pass to the application-defined function. For example, this value could point to a data structure or a memory location.

context

This parameter is ignored; specify `kMPOwningProcessRemoteContext`.

Return Value
The value that your remote procedure callback returned.

Discussion
You use this function to execute code on your application's main task. The *remoteProc* function is scheduled on the application's main run loop and run in the default mode (`kCFRunloopDefaultMode`). If you call this function from your application's main task, the *remoteProc* function is executed immediately in the current mode without blocking the task; otherwise, calling this function blocks the current task until the remote call completes.

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

Declared In
`Multiprocessing.h`

MPSetEvent

Merges event flags into a specified event group.

```
OSStatus MPSetEvent (
   MPEventID event,
   MPEventFlags flags
);
```

Parameters

event
The ID of the event group you want to set.

flags
The flags you want to merge into the event group.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
The flags are logically ORed with the current flags in the event group. This procedure is an atomic operation to ensure that multiple updates do not get lost. If tasks are waiting on this event group, the first waiting task is unblocked.

Note that you can call this function from an interrupt handler.

Also see the function `MPWaitForEvent` (page 1505).

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

Declared In
`Multiprocessing.h`

MPSetExceptionHandler

Sets an exception handler for a task.

```
OSStatus MPSetExceptionHandler (
   MPTaskID task,
   MPQueueID exceptionQ
);
```

Parameters

task

The task to associate with the exception handler.

exceptionQ

The message queue to which an exception message will be sent.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
When an exception handler is set and an exception occurs, the task is suspended and a message is sent to the message queue specified by `exceptionQ`. The message contains the following information:

- The first pointer-sized value contains the ID of the task in which the exception occurred.
- The second pointer-sized value contains the type of exception that occurred. See the header file `MachineExceptions.h` for a listing of exception types.
- The last pointer-sized value is set to `NULL` (reserved for future use).

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

Declared In
`Multiprocessing.h`

MPSetQueueReserve

Reserves space for messages on a specified message queue.

```
OSStatus MPSetQueueReserve (
   MPQueueID queue,
   ItemCount count
);
```

Parameters

queue

The ID of the queue whose messages you want to reserve.

count

The number of messages to reserve.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
`MPNotifyQueue` (page 1492) allocates spaces for messages dynamically; that is, memory to hold the message is allocated for the queue at the time of the call. In most cases this method is both speed and storage efficient. However, it is possible that, due to lack of memory resources, space for the message may not be available at the time of the call; in such cases, `MPNotifyQueue` (page 1492) will return `kInsufficientResourcesErr`.

If you must have guaranteed message delivery, or if you need to call `MPNotifyQueue` (page 1492) from an interrupt handler, you should reserve space on the specified queue by calling `MPSetQueueReserve`. Because such allocated space is reserved for duration of the queue's existence, you should avoid straining internal system resources by reserving messages only when absolutely necessary. Note that if you have reserved messages on a queue, additional space cannot be added dynamically if the number of messages exceeds the number reserved for that queue.

The number of reserved messages is set to `count`, lowering or increasing the current number of reserved messages as required. If `count` is set to zero, no messages are reserved for the queue, and space for messages is allocated dynamically.

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

Declared In
`Multiprocessing.h`

MPSetTaskState

Sets state information for a suspended task.

```
OSStatus MPSetTaskState (
   MPTaskID task,
   MPTaskStateKind kind,
   void *info
);
```

Parameters

task

The task whose state information you want to set.

kind

The kind of state information you want to set. See "Task State Constants" (page 1530) for a listing of possible values. Note that some state information is read-only and cannot be changed using this function.

info

A pointer to a data structure holding the state information you want to set. The format of the data structure varies depending on the state information you want to set. See the header file `MachineExceptions.h` for the formats of the various state information structures.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533). If you specify `kMPTaskState32BitMemoryException` for the state information, this function returns `kMPInsufficientResourcesErr`, since the exception state information is read-only. Attempting to set state information for a running task will also return `kMPInsufficientResourcesErr`.

Discussion
You can use this function to set register contents or exception information for a particular task. However, some state information, such as the exception information (as specified by `kMPTaskState32BitMemoryException`) as well as the `MSR`, `ExceptKind`, `DSISR`, and `DAR` machine registers (specified under `kMPTaskStateMachine`) are read-only. Attempting to set the read-only machine registers will do nothing, while attempting to set the exception information will return an error.

Also see the function `MPExtractTaskState` (page 1488).

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

Declared In
`Multiprocessing.h`

MPSetTaskStorageValue

Sets the storage value for a given index number.

```
OSStatus MPSetTaskStorageValue (
   TaskStorageIndex taskIndex,
   TaskStorageValue value
);
```

Parameters

index
The index number whose storage value you want to set.

value
The value you want to set.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
Typically you use `MPSetTaskStorageValue` to store pointers to task-specific structures or data.

Calling this function from within a task effectively assigns a value in a two-dimensional array cross-referenced by task storage index value and the task ID.

Also see the function `MPGetTaskStorageValue` (page 1490).

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

Declared In
`Multiprocessing.h`

MPSetTaskType

Sets the type of the task.

```
OSStatus MPSetTaskType (
   MPTaskID task,
   OSType taskType
);
```

Return Value
The `noErr` result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
This function does nothing and should not be used.

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

Declared In
`Multiprocessing.h`

MPSetTaskWeight

Assigns a relative weight to a task, indicating how much processor time it should receive compared to other available tasks.

```
OSStatus MPSetTaskWeight (
   MPTaskID task,
   MPTaskWeight weight
);
```

Parameters

task
The ID of the task to which you want to assign a weighting.

weight
The relative weight to assign. This value can range from 1 to 10,000, with the default value being 100.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
The approximate processor share is defined as:

```
weight of the task / total weight of available tasks
```

For a set of ready tasks, the amount of CPU time dedicated to the tasks will be determined by the dynamically computed share. Note that the processor share devoted to tasks may deviate from the suggested weighting if critical tasks require attention. For example, a real-time task (such as a QuickTime movie) may require more than its relative weight of processor time, and the scheduler will adjust proportions accordingly.

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

Declared In
`Multiprocessing.h`

MPSetTimerNotify

Sets the notification information associated with a timer.

```
OSStatus MPSetTimerNotify (
   MPTimerID timerID,
   MPOpaqueID anID,
   void *notifyParam1,
   void *notifyParam2,
   void *notifyParam3
);
```

Parameters

timerID

The ID of the timer whose notification information you want to set.

notificationID

The ID of the notification mechanism to associate with the timer. This value should be the ID of an event group, a message queue, or a semaphore.

notifyParam1

If `anID` specifies an event group, this parameter should contain the flags to set in the event group when the timer expires. If `anID` specifies a message queue, this parameter should contain the first pointer-sized value of the message to be sent to the message queue when the timer expires.

notifyParam2

If `anID` specifies a message queue, this parameter should contain the second pointer-sized value of the message to be sent to the message queue when the timer expires. Pass `NULL` if you don't need this parameter.

notifyParam3

If `anID` specifies a message queue, this parameter should contain the third pointer-sized value of the message sent to the message queue when the timer expires. Pass `NULL` if you don't need this parameter.

Return Value

A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion

When the timer expires, Multiprocessing Services checks the notification ID, and if it is valid, notifies the related notification mechanisms (that is, event groups, queues, or semaphores) you had specified in your `MPSetTimerNotify` (page 1500) calls.

You can specify multiple notification mechanisms by calling this function several times. For example, you can call `MPSetTimerNotify` to specify a message queue and then call it again to specify a semaphore. When the timer expires, a message is sent to the message queue and the appropriate semaphore is signaled. You cannot, however, specify more than one notification per notification mechanism (for example, if you call `MPSetTimerNotify` twice, specifying different messages or message queues in each call, the second call will overwrite the first). Note that if a call to `MPSetTimerNotify` returns an error, any previous calls specifying the same timer are still valid; previously set notifications will still be notified when the timer expires.

You can set the notification information at any time. If the timer is armed, it will modify the notification parameters dynamically. If the timer is disarmed, it will modify the notification parameters to be used for the next `MPArmTimer` (page 1473) call.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Multiprocessing.h`

MPSignalSemaphore

Signals a semaphore.

```
OSStatus MPSignalSemaphore (
   MPSemaphoreID semaphore
);
```

Parameters

semaphore

The ID of the semaphore you want to signal.

Return Value

A result code. See "Multiprocessing Services Result Codes" (page 1533). If the value of the semaphore was already at the maximum, `MPSignalSemaphore` returns `kInsufficientResourcesErr`.

Discussion

If tasks are waiting on the semaphore, the oldest (first queued) task is unblocked so that the corresponding `MPWaitOnSemaphore` (page 1507) call for that task completes. Otherwise, if the value of the semaphore is not already equal to its maximum value, it is incremented by one.

Note that you can call this function from an interrupt handler.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Multiprocessing.h`

MPTaskIsPreemptive

Determines whether a task is preemptively scheduled.

```
Boolean MPTaskIsPreemptive (
   MPTaskID taskID
);
```

Parameters

taskID

The task you want to check. Pass `kMPNoID` or `kInvalidID` if you want to specify the current task.

Return Value

If true, the task is preemptively scheduled. If false, the task is cooperatively scheduled.

Discussion

If you have code that may be called from either cooperative or preemptive tasks, that code can call `MPTaskIsPreemptive` if its actions should differ depending on its execution environment.

Note that you can call this function from an interrupt handler.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Multiprocessing.h`

MPTerminateTask

Terminates an existing task.

```
OSStatus MPTerminateTask (
   MPTaskID task,
   OSStatus terminationStatus
);
```

Parameters

task

The ID of the task you wish to terminate.

terminationStatus

A value of type `OSStatus` indicating termination status. This value is sent to the termination status message queue you specified in `MPCreateTask` (page 1479) in place of the task function's result code.

Return Value

A result code. See "Multiprocessing Services Result Codes" (page 1533). If the task to be terminated is already in the process of termination, `MPTerminateTask` returns `kMPInsufficientResourcesErr`. You do not need to take any additional action if this occurs.

Discussion

You should be very careful when calling `MPTerminateTask`. As defined, this call will asynchronously and abruptly terminate a task, potentially leaving whatever structures or resources it was operating upon in an indeterminate state. Mac OS X exacerbates this problem, as MP tasks can use many more system services that are not expecting client threads to asynchronously terminate, and these services do not take the rather complicated steps necessary to protect against, or recover from, such a situation.

However, there are situations in which calling `MPTerminateTask` is useful and relatively safe. One such situation is when your application or service is quitting and you know that the task you wish to terminate is waiting on an MP synchonization construct (queue, event, semaphore or critical region). While you could do this more cleanly by waking the task and causing it to exit on its own, doing so may not always be practical.

For example, suppose you have several service tasks performing background processing for your application. These service tasks wait on a queue, onto which the application places requests for processing. When the task is done with a request, it notifies another queue, which the application polls. Since the main application task is placing items on the shared queue, and receiving notifications when the requests are done, it can track whether or not there are outstanding requests being processed. If all outstanding requests have, in fact, been processed, it is relatively safe to terminate a task (or all tasks) waiting on the request queue.

You should not assume that the task has completed termination when this call returns; the proper way to synchronize with task termination is to wait on the termination queue (specified in `MPCreateTask` (page 1479)) until a message appears. Because task termination is a multistage activity, it is possible for a preemptive task to attempt to terminate a task that is already undergoing termination. In such cases, `MPTerminateTask` returns `kMPInsufficientResourcesErr`.

Note that Multiprocessing Services resources (event groups, queues, semaphores, and critical regions) owned by a preemptive task are not released when that task terminates. If a task has a critical region locked when it terminates, the critical region remains in the locked state. Multiprocessing Services resources no longer needed should be explicitly deleted by the task that handles the termination message. All Multiprocessing Services resources created by tasks are released when their owning process (that is, the host application) terminates.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Multiprocessing.h`

MPThrowException

Throws an exception to a specified task.

```
OSStatus MPThrowException (
   MPTaskID task,
   MPExceptionKind kind
);
```

Parameters

task

The task to which the exception should be thrown.

kind

The type of exception to give to the task.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533). If the task is already suspended or if the task is not defined to take thrown exceptions, the function returns `kMPInsufficientResourcesErr`.

Discussion
The exception is treated in the same manner as any other exception taken by a task. However, since it is asynchronous, it may not be presented immediately.

By convention, you should set the exception kind to `kMPTaskStoppedErr` if you want to suspend a task. In general, you should do so only if you are debugging and wish to examine the state of the task. Otherwise you should block the task using one of the traditional notification mechanisms (such as a message queue).

An exception can be thrown at any time, whether that task is running, eligible to be run (that is, ready), or blocked. The task is suspended and an exception message may be generated the next time the task is about to run. Note that this may never occur— for example, if the task is deadlocked or the resource it is waiting on is never released. If the task is currently blocked when this function is executed, `kMPTaskBlockedErr` is returned. If the task was suspended immediately at the conclusion of this function call the return value is `kMPTaskStoppedErr`.

In Mac OS X, this function is available but is not implemented.

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

Declared In
`Multiprocessing.h`

MPUnregisterDebugger

Unregisters a debugger.

```
OSStatus MPUnregisterDebugger (
   MPQueueID queue
);
```

Parameters

queue

The ID of the queue whose debugger you want to unregister.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
In Mac OS X, this function is available but is not implemented. Use system debugging services to write a debugger for Mac OS X.

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

Declared In
`Multiprocessing.h`

MPWaitForEvent

Retrieves event flags from a specified event group.

```
OSStatus MPWaitForEvent (
   MPEventID event,
   MPEventFlags *flags,
   Duration timeout
);
```

Parameters

event

The event group whose flags you want to retrieve.

flags

On return, `flags` contains the flags of the specified event group. Pass `NULL` if you do not need any flag information.

timeout

The maximum time to wait for events before timing out. See "Timer Duration Constants" (page 1531) for a list of constants you can use to specify the wait interval.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
This function obtains event flags from the specified event group. The timeout specifies how long to wait for events if none are present when the call is made. If any flags are set when this function is called, all the flags in the event group are moved to the `flag` field and the event group is cleared. This obtaining and clearing action is an atomic operation to ensure that no updates are lost. If multiple tasks are waiting on an event group, only one can obtain any particular set of flags.

If you call this function from a cooperative task, you should specify only `kDurationImmediate` for the timeout length; other waits will cause the task to block.

Also see the function `MPSetEvent` (page 1496).

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

Declared In
`Multiprocessing.h`

MPWaitOnQueue

Obtains a message from a specified message queue.

```
OSStatus MPWaitOnQueue (
   MPQueueID queue,
   void **param1,
   void **param2,
   void **param3,
   Duration timeout
);
```

Parameters

queue
The ID of the message queue from which to receive the notification.

param1
On return, the first pointer-sized value of the notification message. Pass `NULL` if you do not need this portion of the message.

param2
On return, the second pointer-sized value of the notification message. Pass `NULL` if you do not need this portion of the message.

param3
On return, the third pointer-sized value of the notification message. Pass `NULL` if you do not need this portion of the message.

timeout
The time to wait for a notification before timing out. See "Timer Duration Constants" (page 1531) for a list of constants you can use to specify the wait interval.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
This function receives a message from the specified message queue. If no messages are currently available, the timeout specifies how long the function should wait for one. Tasks waiting on the queue are handled in a first in, first out manner; that is, the first task to wait on the queue receives the message from the `MPNotifyQueue` (page 1492) call.

After calling this function, when a message appears, it is removed from the queue and the three fields, `param1`, `param2`, and `param3` are set to the values specified by the message text. Note these parameters are pointers to variables to be set with the message text.

If you call this function from a cooperative task, you should specify only `kDurationImmediate` for the timeout length; other waits will cause the task to block.

Also see the function `MPNotifyQueue` (page 1492).

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

Declared In
`Multiprocessing.h`

MPWaitOnSemaphore

Waits on a semaphore

```
OSStatus MPWaitOnSemaphore (
   MPSemaphoreID semaphore,
   Duration timeout
);
```

Parameters

semaphore
The ID of the semaphore you want to wait on.

timeout
The maximum time the function should wait before timing out. See "Timer Duration Constants" (page 1531) for a list of constants you can use to specify the wait interval.

Return Value
A result code. See "Multiprocessing Services Result Codes" (page 1533).

Discussion
If the value of the semaphore is greater than zero, the value is decremented and the function returns with `noErr`. Otherwise, the task is blocked awaiting a signal until the specified timeout is exceeded.

If you call this function from a cooperative task, you should specify only `kDurationImmediate` for the timeout length; other waits will cause the task to block.

Also see the function `MPSignalSemaphore` (page 1502).

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

Declared In
`Multiprocessing.h`

MPYield

Allows a task to yield the processor to another task.

```
void MPYield (
   void
);
```

Discussion
This function indicates to the scheduler that another task can run. Other than possibly yielding the processor to another task or application, the call has no effect. Note that since tasks are preemptively scheduled, an implicit yield may occur at any point, whether or not this function is called.

In most cases you should not need to call this function. The most common use of `MPYield` is to release the processor when a task is in a loop in which further progress is dependent on other tasks, and the task cannot be blocked by waiting on a Multiprocessing Services resource. You should avoid such busy waiting whenever possible.

Note that you can call this function from an interrupt handler.

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

Declared In
`Multiprocessing.h`

_MPIsFullyInitialized

Indicates whether Multiprocessing Services is available for use.

```
Boolean _MPIsFullyInitialized (
   void
);
```

Return Value
If true, Multiprocessing Services is available for use; otherwise, false.

Declared In
`Multiprocessing.h`

Callbacks

MPRemoteProcedure

Defines a remote procedure call.

```
typedef void* (*MPRemoteProcedure) (
    void *parameter
);
```

For example, this is how you would declare the application-defined function if you were to name the function `MyRemoteProcedure`:

```
void* MyRemoteProcedure (
    void *parameter
);
```

Parameters

parameter
A pointer to the application-defined value you passed to the function `MPRemoteCallCFM` (page 1495). For example, this value could point to a data structure or a memory location.

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

Declared In
`Multiprocessing.h`

TaskProc

Defines the entry point of a task.

```
typedef OSStatus (*TaskProc) (
    void *parameter
);
```

For example, this is how you would declare the application-defined function if you were to name the function `MyTaskProc`:

```
OSStatus MyTaskProc (
    void *parameter
);
```

Parameters

parameter

A pointer to the application-defined value you passed to the function `MPCreateTask` (page 1479). For example, this value could point to a data structure or a memory location.

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

Declared In
`Multiprocessing.h`

Data Types

MPAddressSpaceID

```
typedef struct OpaqueMPAddressSpaceID * MPAddressSpaceID;
```

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

Declared In
`Multiprocessing.h`

MPAddressSpaceInfo

```
struct MPAddressSpaceInfo {
    PBVersion version;
    MPProcessID processID;
    MPCoherenceID groupID;
    ItemCount nTasks;
    UInt32 vsid[16];
};
typedef struct MPAddressSpaceInfo MPAddressSpaceInfo;
```

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

Declared In
`MultiprocessingInfo.h`

MPAreaID

```
typedef struct OpaqueMPAreaID * MPAreaID;
```

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

Declared In
`Multiprocessing.h`

MPCoherenceID

Represents a memory coherence group.

```
typedef struct OpaqueMPCoherenceID * MPCoherenceID;
```

Discussion
A coherence group is the set of processors and other bus controllers that have cache-coherent access to memory. Mac OS 9 defines only one coherence group, which is all the processors that can access internal memory (RAM). Other coherence groups are possible; for example, a PCI card with its own memory and processors can comprise a coherence group.

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

Declared In
`Multiprocessing.h`

MPConsoleID

```
typedef struct OpaqueMPConsoleID * MPConsoleID;
```

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

Declared In
Multiprocessing.h

MPCpuID

Represents a CPU ID.

```
typedef struct OpaqueMPCpuID * MPCpuID;
```

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

Declared In
Multiprocessing.h

MPCriticalRegionID

Represents a critical region ID, which Multiprocessing Services uses to manipulate critical regions.

```
typedef struct OpaqueMPCriticalRegionID * MPCriticalRegionID;
```

Discussion
You obtain a critical region ID by calling the function MPCreateCriticalRegion (page 1476).

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

Declared In
Multiprocessing.h

MPCriticalRegionInfo

```
struct MPCriticalRegionInfo {
    PBVersion version;
    MPProcessID processID;
    OSType regionName;
    ItemCount nWaiting;
    MPTaskID waitingTaskID;
    MPTaskID owningTask;
    ItemCount count;
};
typedef struct MPCriticalRegionInfo MPCriticalRegionInfo;
```

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

Declared In
MultiprocessingInfo.h

MPEventFlags

Represents event information for an event group.

```
typedef UInt32 MPEventFlags;
```

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

Declared In
`Multiprocessing.h`

MPEventID

Represents an event group ID, which Multiprocessing Services uses to manipulate event groups.

```
typedef struct OpaqueMPEventID * MPEventID;
```

Discussion
You obtain an event group ID by calling the function `MPCreateEvent` (page 1477).

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

Declared In
`Multiprocessing.h`

MPEventInfo

```
struct MPEventInfo {
    PBVersion version;
    MPProcessID processID;
    OSType eventName;
    ItemCount nWaiting;
    MPTaskID waitingTaskID;
    MPEventFlags events;
};
typedef struct MPEventInfo MPEventInfo;
```

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

Declared In
`MultiprocessingInfo.h`

MPExceptionKind

Represents the kind of exception thrown.

```
typedef UInt32 MPExceptionKind;
```

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

Declared In
Multiprocessing.h

MPNotificationID

Represents a notification ID, which Multiprocessing Services uses to manipulate kernel notifications.

```
typedef struct OpaqueMPNotificationID * MPNotificationID;
```

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

Declared In
Multiprocessing.h

MPNotificationInfo

```
struct MPNotificationInfo {
    PBVersion version;
    MPProcessID processID;
    OSType notificationName;
    MPQueueID queueID;
    void * p1;
    void * p2;
    void * p3;
    MPEventID eventID;
    MPEventFlags events;
    MPSemaphoreID semaphoreID;
};
typedef struct MPNotificationInfo MPNotificationInfo;
```

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

Declared In
MultiprocessingInfo.h

MPOpaqueID

Represents a generic notification ID (that is, an ID that could be a queue ID, event ID, kernel notification ID, or semaphore ID).

```
typedef struct OpaqueMPOpaqueID * MPOpaqueID;
```

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

Declared In
Multiprocessing.h

MPOpaqueIDClass

```
typedef UInt32 MPOpaqueIDClass;
```

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

Declared In
`Multiprocessing.h`

MPPageSizeClass

```
typedef UInt32 MPPageSizeClass;
```

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

Declared In
`Multiprocessing.h`

MPProcessID

Represents a process ID.

```
typedef struct OpaqueMPProcessID * MPProcessID;
```

Discussion
Note that this process ID is identical to the process ID (or context ID) handled by the Code Fragment Manager.

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

Declared In
`Multiprocessing.h`

MPQueueID

Represents a queue ID, which Multiprocessing Services uses to manipulate message queues.

```
typedef struct OpaqueMPQueueID * MPQueueID;
```

Discussion
You obtain a queue ID by calling the function `MPCreateQueue` (page 1478).

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

Declared In
`Multiprocessing.h`

MPQueueInfo

```
struct MPQueueInfo {
    PBVersion version;
    MPProcessID processID;
    OSType queueName;
    ItemCount nWaiting;
    MPTaskID waitingTaskID;
    ItemCount nMessages;
    ItemCount nReserved;
    void * p1;
    void * p2;
    void * p3;
};
typedef struct MPQueueInfo MPQueueInfo;
```

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

Declared In
`MultiprocessingInfo.h`

MPSemaphoreCount

Represents a semaphore count.

```
typedef ItemCount MPSemaphoreCount;
```

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

Declared In
`Multiprocessing.h`

MPSemaphoreID

Represents a semaphore ID, which Multiprocessing Services uses to manipulate semaphores.

```
typedef struct OpaqueMPSemaphoreID * MPSemaphoreID;
```

Discussion
You obtain a semaphore ID by calling the function `MPCreateSemaphore` (page 1478).

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

Declared In
`Multiprocessing.h`

MPSemaphoreInfo

```
struct MPSemaphoreInfo {
    PBVersion version;
    MPProcessID processID;
    OSType semaphoreName;
    ItemCount nWaiting;
    MPTaskID waitingTaskID;
    ItemCount maximum;
    ItemCount count;
};
typedef struct MPSemaphoreInfo MPSemaphoreInfo;
```

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

Declared In
`MultiprocessingInfo.h`

MPTaskID

Represents a task ID.

```
typedef struct OpaqueMPTaskID * MPTaskID;
```

Discussion
You obtain a task ID by calling the function `MPCreateTask` (page 1479).

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

Declared In
`Multiprocessing.h`

MPTaskInfo

Contains information about a task.

```
struct MPTaskInfo {
    PBVersion version;
    OSType name;
    OSType queueName;
    UInt16 runState;
    UInt16 lastCPU;
    UInt32 weight;
    MPProcessID processID;
    AbsoluteTime cpuTime;
    AbsoluteTime schedTime;
    AbsoluteTime creationTime;
    ItemCount codePageFaults;
    ItemCount dataPageFaults;
    ItemCount preemptions;
    MPCpuID cpuID;
    MPOpaqueID blockedObject;
    MPAddressSpaceID spaceID;
    LogicalAddress stackBase;
    LogicalAddress stackLimit;
    LogicalAddress stackCurr;
};
typedef struct MPTaskInfo MPTaskInfo;
```

Fields

`version`
: The version of this data structure.

`name`
: The name of the task.

`queueName`
: A four-byte code indicating the status of the queue waiting on the task.

`runState`
: The current state of the task (running, ready, or blocked).

`lastCPU`
: The address of the last processor that ran this task.

`weight`
: The weighting assigned to this task.

`processID`
: The ID of the process that owns this task.

`cpuTime`
: The accumulated CPU time used by the task.

`schedTime`
: The time when the task was last scheduled.

`creationTime`
: The time when the task was created.

`codePageFaults`
: The number of page faults that occurred during code execution.

`dataPageFaults`
: The number of page faults that occurred during data access.

`preemptions`
: The number of times this task was preempted.

`cpuID`
The ID of the last processor that ran this task.

`blockedObject`
Reserved for use by Mac OS X.

`spaceID`
Address space ID of this task.

`stackBase`
The lowest memory address of the task's stack.

`stackLimit`
The highest memory address of the task's stack.

`stackCurr`
The current stack address.

Discussion
If you specify the `kMPTaskStateTaskInfo` constant when calling the function `MPExtractTaskState` (page 1488) , Multiprocessing Services returns state information in an `MPTaskInfo` structure.

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

Declared In
`Multiprocessing.h`

MPTaskInfoVersion2

```
struct MPTaskInfoVersion2 {
    PBVersion version;
    OSType name;
    OSType queueName;
    UInt16 runState;
    UInt16 lastCPU;
    UInt32 weight;
    MPProcessID processID;
    AbsoluteTime cpuTime;
    AbsoluteTime schedTime;
    AbsoluteTime creationTime;
    ItemCount codePageFaults;
    ItemCount dataPageFaults;
    ItemCount preemptions;
    MPCpuID cpuID;
};
typedef struct MPTaskInfoVersion2 MPTaskInfoVersion2;
```

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

Declared In
`Multiprocessing.h`

MPTaskStateKind

```
typedef UInt32 MPTaskStateKind;
```

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

Declared In
`Multiprocessing.h`

MPTaskWeight

Represents the relative processor weighting of a task.

```
typedef UInt32 MPTaskWeight;
```

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

Declared In
`Multiprocessing.h`

MPTimerID

Represents a timer ID.

```
typedef struct OpaqueMPTimerID * MPTimerID;
```

Discussion
You obtain a timer ID by calling the function `MPCreateTimer` (page 1480).

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

Declared In
`Multiprocessing.h`

TaskStorageIndex

Represents a task storage index value used by functions described in "Accessing Per-Task Storage Variables."

```
typedef ItemCount TaskStorageIndex;
```

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

Declared In
`Multiprocessing.h`

TaskStorageValue

Represents a task storage value used by functions described in "Accessing Per-Task Storage Variables."

```
typedef LogicalAddress TaskStorageValue;
```

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

Declared In
`Multiprocessing.h`

Constants

Allocation constants

The maximum memory allocation size.

```
enum {
    kMPMaxAllocSize = 1024L * 1024 * 1024
};
```

Constants
`kMPMaxAllocSize`
The maximum allocation size: 1GB.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

Task IDs

Use to specify no task ID.

```
enum {
    kMPNoID = kInvalidID
};
```

Constants
`kMPNoID`
No task ID.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

Discussion
Used when calling `MPTaskIsPreemptive` (page 1502) if you want to specify the current task.

Data Structure Version Constants

Data structure version information constants.

```
enum {
    kMPQueueInfoVersion = 1L | (kOpaqueQueueID << 16),
    kMPSemaphoreInfoVersion = 1L | (kOpaqueSemaphoreID << 16),
    kMPEventInfoVersion = 1L | (kOpaqueEventID << 16),
    kMPCriticalRegionInfoVersion = 1L | (kOpaqueCriticalRegionID << 16),
    kMPNotificationInfoVersion = 1L | (kOpaqueNotificationID << 16),
    kMPAddressSpaceInfoVersion = 1L | (kOpaqueAddressSpaceID << 16)
};
```

Constants

`kMPQueueInfoVersion`
The `MPQueueInfo` structure version.
Available in Mac OS X v10.0 and later.
Declared in `MultiprocessingInfo.h`.

`kMPSemaphoreInfoVersion`
The `MPSemaphoreInfo` structure version.
Available in Mac OS X v10.0 and later.
Declared in `MultiprocessingInfo.h`.

`kMPEventInfoVersion`
The `MPEventInfo` structure version.
Available in Mac OS X v10.0 and later.
Declared in `MultiprocessingInfo.h`.

`kMPCriticalRegionInfoVersion`
The `MPCriticalRegionInfo` structure version.
Available in Mac OS X v10.0 and later.
Declared in `MultiprocessingInfo.h`.

`kMPNotificationInfoVersion`
The `MPNotificationInfo` structure version.
Available in Mac OS X v10.0 and later.
Declared in `MultiprocessingInfo.h`.

`kMPAddressSpaceInfoVersion`
The `MPAddressSpaceInfo` structure version.
Available in Mac OS X v10.1 and later.
Declared in `MultiprocessingInfo.h`.

Values for the MPOpaqueIDClass type

Constants indicating the source of a generic notification.

```
enum {
    kOpaqueAnyID = 0,
    kOpaqueProcessID = 1,
    kOpaqueTaskID = 2,
    kOpaqueTimerID = 3,
    kOpaqueQueueID = 4,
    kOpaqueSemaphoreID = 5,
    kOpaqueCriticalRegionID = 6,
    kOpaqueCpuID = 7,
    kOpaqueAddressSpaceID = 8,
    kOpaqueEventID = 9,
    kOpaqueCoherenceID = 10,
    kOpaqueAreaID = 11,
    kOpaqueNotificationID = 12,
    kOpaqueConsoleID = 13
};
```

Constants

`kOpaqueAnyID`
Any source.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kOpaqueProcessID`
A process.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kOpaqueTaskID`
A task.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kOpaqueTimerID`
A timer.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kOpaqueQueueID`
A queue.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kOpaqueSemaphoreID`
A semaphore.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kOpaqueCriticalRegionID`
A critical region.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kOpaqueCpuID`
A CPU.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kOpaqueAddressSpaceID`
An address space.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kOpaqueEventID`
An event.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kOpaqueCoherenceID`
A coherence group.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kOpaqueAreaID`
An area.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kOpaqueNotificationID`
A notification.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kOpaqueConsoleID`
A console.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

Memory Allocation Alignment Constants

Specify the alignment of the desired memory block when calling the `MPAllocateAligned` function.

```
enum {
    kMPAllocateDefaultAligned = 0,
    kMPAllocate8ByteAligned = 3,
    kMPAllocate16ByteAligned = 4,
    kMPAllocate32ByteAligned = 5,
    kMPAllocate1024ByteAligned = 10,
    kMPAllocate4096ByteAligned = 12,
    kMPAllocateMaxAlignment = 16,
    kMPAllocateAltiVecAligned = kMPAllocate16ByteAligned,
    kMPAllocateVMXAligned = kMPAllocateAltiVecAligned,
    kMPAllocateVMPageAligned = 254,
    kMPAllocateInterlockAligned = 255
};
```

Constants

`kMPAllocateDefaultAligned`
Use the default alignment.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kMPAllocate8ByteAligned`
Use 8-byte alignment.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kMPAllocate16ByteAligned`
Use 16-byte alignment.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kMPAllocate32ByteAligned`
Use 32-byte alignment.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kMPAllocate1024ByteAligned`
Use 1024-byte alignment.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kMPAllocate4096ByteAligned`
Use 4096-byte alignment.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kMPAllocateMaxAlignment`
Use the maximum alignment (65536 byte).
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kMPAllocateAltiVecAligned`
Use AltiVec alignment.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kMPAllocateVMXAligned`
Use VMX (now called AltiVec) alignment.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPAllocateVMPageAligned`
Use virtual memory page alignment. This alignment is set at runtime.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPAllocateInterlockAligned`
Use interlock alignment, which is the alignment needed to allow the use of CPU interlock instructions (that is, `lwarx` and `stwcx`.) on the returned memory address. This alignment is set at runtime. In most cases you would never need to use this alignment.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

Memory Allocation Option Constants

Specify optional actions when calling the `MPAllocateAligned` function.

```
enum {
    kMPAllocateClearMask = 0x0001,
    kMPAllocateGloballyMask = 0x0002,
    kMPAllocateResidentMask = 0x0004,
    kMPAllocateNoGrowthMask = 0x0010,
    kMPAllocateNoCreateMask = 0x0020
};
```

Constants

`kMPAllocateClearMask`
Zero out the allocated memory block.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPAllocateGloballyMask`
Allocate memory from in memory space that is visible to all processes. Note that such globally-allocated space is not automatically reclaimed when the allocating process terminates. By default, `MPAllocateAligned` (page 1472) allocates memory from process-specific (that is, not global) memory.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPAllocateResidentMask`
Allocate memory from resident memory only (that is, the allocated memory is not pageable).

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPAllocateNoGrowthMask`
Do not attempt to grow the pool of available memory. Specifying this option is useful, as attempting to grow memory may cause your task to block until such memory becomes available.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPAllocateNoCreateMask`
Do not attempt to create the pool if it does not yet exist.

Available in Mac OS X v10.1 and later.

Declared in `Multiprocessing.h`.

MPDebuggerLevel

Indicates the debugger level.

```
typedef UInt32 MPDebuggerLevel;
enum {
    kMPLowLevelDebugger = 0x00000000,
    kMPMidLevelDebugger = 0x10000000,
    kMPHighLevelDebugger = 0x20000000
};
```

Constants

`kMPLowLevelDebugger`
The low-level debugger.

Available in Mac OS X v10.1 and later.

Declared in `Multiprocessing.h`.

`kMPMidLevelDebugger`
The mid-level debugger.

Available in Mac OS X v10.1 and later.

Declared in `Multiprocessing.h`.

`kMPHighLevelDebugger`
The high-level debugger.

Available in Mac OS X v10.1 and later.

Declared in `Multiprocessing.h`.

Library Version Constants

Identifies the current library version.

```
enum {
    MPLibrary_MajorVersion = 2,
    MPLibrary_MinorVersion = 3,
    MPLibrary_Release = 1,
    MPLibrary_DevelopmentRevision = 1
};
```

Constants

`MPLibrary_MajorVersion`
Major version number.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`MPLibrary_MinorVersion`
Minor version number.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`MPLibrary_Release`
Release number.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`MPLibrary_DevelopmentRevision`
Development revision number.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

Remote Call Context Option Constants

Specify which contexts are allowed to execute the callback function when using `MPRemoteCall`.

```
enum {
    kMPAnyRemoteContext = 0,
    kMPOwningProcessRemoteContext = 1,
    kMPInterruptRemoteContext = 2,
    kMPAsyncInterruptRemoteContext = 3
};
typedef UInt8 MPRemoteContext;
```

Constants

`kMPAnyRemoteContext`
Any cooperative context can execute the function. Note that the called function may not have access to any of the owning context's process-specific low-memory values.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPOwningProcessRemoteContext`
Only the context that owns the task can execute the function.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPInterruptRemoteContext`
Unsupported in Mac OS X.

Available in Mac OS X v10.1 and later.

Declared in `Multiprocessing.h`.

`kMPAsyncInterruptRemoteContext`
Unsupported in Mac OS X.

Available in Mac OS X v10.1 and later.

Declared in `Multiprocessing.h`.

Discussion
These constants are used to support older versions of Mac OS and are ignored in Mac OS X.

Task Creation Options

Specify optional actions when calling the `MPCreateTask` function.

```
enum {
    kMPCreateTaskSuspendedMask = 1L << 0,
    kMPCreateTaskTakesAllExceptionsMask = 1L << 1,
    kMPCreateTaskNotDebuggableMask = 1L << 2,
    kMPCreateTaskValidOptionsMask = kMPCreateTaskSuspendedMask |
kMPCreateTaskTakesAllExceptionsMask | kMPCreateTaskNotDebuggableMask
};
typedef OptionBits MPTaskOptions;
```

Constants

`kMPCreateTaskSuspendedMask`

Unsupported in Mac OS X.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPCreateTaskTakesAllExceptionsMask`

The task will take all exceptions, including those normally handled by the system, such as page faults.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPCreateTaskNotDebuggableMask`

Unsupported in Mac OS X.

Available in Mac OS X v10.1 and later.

Declared in `Multiprocessing.h`.

`kMPCreateTaskValidOptionsMask`

Include all valid options for this task.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

Task Exception Disposal Constants

Specify actions to take on an exception when passed in the `action` parameter of the `MPDisposeTaskException` function.

```
enum {
    kMPTaskPropagate = 0,
    kMPTaskResumeStep = 1,
    kMPTaskResumeBranch = 2,
    kMPTaskResumeMask = 0x0000,
    kMPTaskPropagateMask = 1 << kMPTaskPropagate,
    kMPTaskResumeStepMask = 1 << kMPTaskResumeStep,
    kMPTaskResumeBranchMask = 1 << kMPTaskResumeBranch
};
```

Constants

`kMPTaskPropagate`
The exception is propagated.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kMPTaskResumeStep`
The task is resumed and single step is enabled.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kMPTaskResumeBranch`
The task is resumed and branch stepping is enabled.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kMPTaskResumeMask`
Resume the task.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kMPTaskPropagateMask`
Propagate the exception to the next debugger level.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kMPTaskResumeStepMask`
Resume the task and enable single stepping.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

`kMPTaskResumeBranchMask`
Resume the task and enable branch stepping.
Available in Mac OS X v10.0 and later.
Declared in `Multiprocessing.h`.

Task Information Structure Version Constant

Indicates the current version of the `MPTaskInfo` structure (returned as the first field).

```
enum {
    kMPTaskInfoVersion = 3
};
```

Constants

`kMPTaskInfoVersion`

The current version of the task information structure.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

Task Run State Constants

Indicate the state of the task when returned as part of the `MPTaskInfo` data structure.

```
enum {
    kMPTaskBlocked = 0,
    kMPTaskReady = 1,
    kMPTaskRunning = 2
};
```

Constants

`kMPTaskBlocked`

The task is blocked..

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPTaskReady`

The task is ready for execution.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPTaskRunning`

The task is currently running.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

Task State Constants

Specify what states you want to set or obtain when calling the `MPExtractTaskState` or `MPSetTaskState` functions.

```
enum {
    kMPTaskStateRegisters = 0,
    kMPTaskStateFPU = 1,
    kMPTaskStateVectors = 2,
    kMPTaskStateMachine = 3,
    kMPTaskState32BitMemoryException = 4,
    kMPTaskStateTaskInfo = 5
};
```

Constants

`kMPTaskStateRegisters`
The task's general-purpose (GP) registers. The `RegisterInformationPowerPC` structure in `MachineExceptions.h` defines the format of this information.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPTaskStateFPU`
The task's floating point registers. The `FPUInformationPowerPC` structure in `MachineExceptions.h` defines the format of this information.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPTaskStateVectors`
The task's vector registers. The `VectorInformationPowerPC` structure in `MachineExceptions.h` defines the format of this information.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPTaskStateMachine`
The task's machine registers. The `MachineInformationPowerPC` structure in `MachineExceptions.h` defines the format of this information. Note that the MSR, ExceptKind, DSISR, and DAR registers are read-only.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPTaskState32BitMemoryException`
The task's exception information for older 32-bit memory exceptions (that is, memory exceptions on 32-bit CPUs). The `MemoryExceptionInformation` structure in `MachineExceptions.h` defines the format of this information. This exception information is read-only.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPTaskStateTaskInfo`
Static and dynamic information about the task, as described by the data structure `MPTaskInfo` (page 1516). This task information is read-only.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

Timer Duration Constants

Specify the maximum time a task should wait for an event to occur.

```
enum {
    kDurationImmediate = 0,
    kDurationForever = 0x7FFFFFFF,
    kDurationMillisecond = 1,
    kDurationMicrosecond = -1
};
```

Constants

`kDurationImmediate`

The task times out immediately, whether or not the event has occurred. If the event occurred, the return status is `noErr`. If the event did not occur, the return status is `kMPTimeoutErr` (assuming no other errors occurred).

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kDurationForever`

The task waits forever. The blocking call waits until either the event occurs, or until the object being waited upon (such as a message queue) is deleted.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kDurationMillisecond`

The task waits one millisecond before timing out.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kDurationMicrosecond`

The task waits one microsecond before timing out.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

Discussion

You can use these constants in conjunction with other values to indicate specific wait intervals. For example, to wait 1 second, you can pass `kDurationMillisecond * 1000`.

Timer Option Masks

Indicate optional actions when calling `MPArmTimer`.

```
enum {
    kMPPreserveTimerIDMask = 1L << 0,
    kMPTimeIsDeltaMask = 1L << 1,
    kMPTimeIsDurationMask = 1L << 2
};
```

Constants

`kMPPreserveTimerIDMask`

Specifying this mask prevents the timer from being deleted when it expires.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPTimeIsDeltaMask`

Specifying this mask indicates that the specified time should be added to the previous expiration time to form the new expiration time. You can use this mask to compensate for timing drift caused by the finite amount of time required to arm the timer, receive the notification, and so on.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

`kMPTimeIsDurationMask`

Specifying this mask indicates that the specified expiration time is of type `Duration`. You can use this mask to avoid having to call time conversion routines when specifying an expiration time.

Available in Mac OS X v10.0 and later.

Declared in `Multiprocessing.h`.

Result Codes

Result codes defined for Multiprocessing Services are listed below.

Result Code	Value	Description
`kMPIterationEndErr`	-29275	Available in Mac OS X v10.0 and later.
`kMPPrivilegedErr`	-29276	Available in Mac OS X v10.0 and later.
`kMPProcessCreatedErr`	-29288	Available in Mac OS X v10.0 and later.
`kMPProcessTerminatedErr`	-29289	Available in Mac OS X v10.0 and later.
`kMPTaskCreatedErr`	-29290	Available in Mac OS X v10.0 and later.
`kMPTaskBlockedErr`	-29291	The desired task is blocked. Available in Mac OS X v10.0 and later.
`kMPTaskStoppedErr`	-29292	The desired task is stopped. Available in Mac OS X v10.0 and later.
`kMPDeletedErr`	-29295	The desired notification the function was waiting upon was deleted. Available in Mac OS X v10.0 and later.
`kMPTimeoutErr`	-29296	The designated timeout interval passed before the function could take action. Available in Mac OS X v10.0 and later.
`kMPInsufficientResourcesErr`	-29298	Could not complete task due to unavailable Multiprocessing Services resources. Note that many functions return this value as a general error when the desired action could not be performed. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`kMPInvalidIDErr`	-29299	Invalid ID value. For example, an invalid message queue ID was passed to `MPNotifyQueue`. Available in Mac OS X v10.0 and later.

Gestalt Constants

You can determine which system software calls are preemptively-safe for Multiprocessing Services by using the preemptive function attribute selectors defined in the Gestalt Manager. For more information, see *Gestalt Manager Reference*.

Pascal String Utilities Reference

Framework:	CoreServices/CoreServices.h
Declared in	MC68000Test.h ddrt.h Disassembler68k.h PLStringFuncs.h IntEnv.h MacRuntime.h MPWLibsDebug.h

Overview

Pascal String Utilities is an API that provides functions for performing common string manipulations, such as concatenation and copying, on Pascal strings. Although Unicode is the preferred encoding for strings on Mac OS X, you may find these functions useful if your application handles Pascal strings as well.

This category also includes structures and constants defining the PEF binary storage format.

Carbon fully supports the functions that assist you in manipulating Pascal strings.

Important: Pascal String Utilities is deprecated as of Mac OS X v10.4. You should update your applications to use Core Foundation Strings (CFStrings) instead. If you need to convert Pascal strings, you can use functions like `CFStringCreateWithPascalString` to do so.

Functions

PLpos

(Deprecated in Mac OS X v10.4. Use Core Foundation strings (CFStrings) instead. See *CFString Reference*.)

```
short PLpos (
   ConstStr255Param str1,
   ConstStr255Param searchStr
);
```

Parameters

str1

str2

Return Value

Discussion

Special Considerations

If you need to convert Pascal strings, you can use functions like `CFStringCreateWithPascalString` to do so.

Version Notes

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PLStringFuncs.h`

PLstrcat

(Deprecated in Mac OS X v10.4. Use Core Foundation strings (CFStrings) instead. See *CFString Reference*.)

```
StringPtr PLstrcat (
   StringPtr str,
   ConstStr255Param append
);
```

Parameters

str1

str2

Return Value

Discussion

Special Considerations

If you need to convert Pascal strings, you can use functions like `CFStringCreateWithPascalString` to do so.

Version Notes

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PLStringFuncs.h`

PLstrchr

(Deprecated in Mac OS X v10.4. Use Core Foundation strings (CFStrings) instead. See *CFString Reference*.)

```
Ptr PLstrchr (
   ConstStr255Param str1,
   short ch1
);
```

Parameters

str1

Return Value

Discussion

Special Considerations

If you need to convert Pascal strings, you can use functions like `CFStringCreateWithPascalString` to do so.

Version Notes

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`PLStringFuncs.h`

PLstrcmp

(Deprecated in Mac OS X v10.4. Use Core Foundation strings (CFStrings) instead. See *CFString Reference*.)

```
short PLstrcmp (
   ConstStr255Param str1,
   ConstStr255Param str2
);
```

Parameters

str1

str2

Return Value

Discussion

Special Considerations

If you need to convert Pascal strings, you can use functions like `CFStringCreateWithPascalString` to do so.

Version Notes

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
`PLStringFuncs.h`

PLstrcpy

(Deprecated in Mac OS X v10.4. Use Core Foundation strings (CFStrings) instead. See *CFString Reference*.)

```
StringPtr PLstrcpy (
   StringPtr dest,
   ConstStr255Param source
);
```

Parameters

str1

str2

Return Value

Discussion

Special Considerations

If you need to convert Pascal strings, you can use functions like `CFStringCreateWithPascalString` to do so.

Version Notes

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Related Sample Code
SoftVDigX

Declared In
`PLStringFuncs.h`

PLstrlen

(Deprecated in Mac OS X v10.4. Use Core Foundation strings (CFStrings) instead. See *CFString Reference*.)

```
short PLstrlen (
   ConstStr255Param str
);
```

Parameters

str

Return Value

Discussion

Special Considerations

If you need to convert Pascal strings, you can use functions like `CFStringCreateWithPascalString` to do so.

Version Notes

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`PLStringFuncs.h`

PLstrncat

(Deprecated in Mac OS X v10.4. Use Core Foundation strings (CFStrings) instead. See *CFString Reference*.)

```
StringPtr PLstrncat (
   StringPtr str1,
   ConstStr255Param append,
   short num
);
```

Parameters

str1

str2

Return Value

Discussion

Special Considerations

If you need to convert Pascal strings, you can use functions like `CFStringCreateWithPascalString` to do so.

Version Notes

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`PLStringFuncs.h`

PLstrncmp

(Deprecated in Mac OS X v10.4. Use Core Foundation strings (CFStrings) instead. See *CFString Reference*.)

```
short PLstrncmp (
   ConstStr255Param str1,
   ConstStr255Param str2,
   short num
);
```

Parameters

str1

str2

Return Value

Discussion

Special Considerations

If you need to convert Pascal strings, you can use functions like `CFStringCreateWithPascalString` to do so.

Version Notes

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PLStringFuncs.h`

PLstrncpy

(Deprecated in Mac OS X v10.4. Use Core Foundation strings (CFStrings) instead. See *CFString Reference*.)

```
StringPtr PLstrncpy (
   StringPtr dest,
   ConstStr255Param source,
   short num
);
```

Parameters

str1

str2

Return Value

Discussion

Special Considerations

If you need to convert Pascal strings, you can use functions like `CFStringCreateWithPascalString` to do so.

Version Notes

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PLStringFuncs.h`

PLstrpbrk

(Deprecated in Mac OS X v10.4. Use Core Foundation strings (CFStrings) instead. See *CFString Reference*.)

```
Ptr PLstrpbrk (
   ConstStr255Param str1,
   ConstStr255Param charSet
);
```

Parameters

str1

str2

Return Value

Discussion

Special Considerations

If you need to convert Pascal strings, you can use functions like `CFStringCreateWithPascalString` to do so.

Version Notes

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PLStringFuncs.h`

PLstrrchr

(Deprecated in Mac OS X v10.4. Use Core Foundation strings (CFStrings) instead. See *CFString Reference*.)

```
Ptr PLstrrchr (
   ConstStr255Param str1,
   short ch1
);
```

Parameters

str1

Return Value

Discussion

Special Considerations

If you need to convert Pascal strings, you can use functions like `CFStringCreateWithPascalString` to do so.

Version Notes

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`PLStringFuncs.h`

PLstrspn

(Deprecated in Mac OS X v10.4. Use Core Foundation strings (CFStrings) instead. See *CFString Reference*.)

```
short PLstrspn (
   ConstStr255Param str1,
   ConstStr255Param charSet
);
```

Parameters

str1

str2

Return Value

Discussion

Special Considerations

If you need to convert Pascal strings, you can use functions like `CFStringCreateWithPascalString` to do so.

Version Notes

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`PLStringFuncs.h`

PLstrstr

(Deprecated in Mac OS X v10.4. Use Core Foundation strings (CFStrings) instead. See *CFString Reference*.)

```
Ptr PLstrstr (
   ConstStr255Param str1,
   ConstStr255Param searchStr
);
```

Parameters

str1

str2

Return Value

Discussion

Special Considerations

If you need to convert Pascal strings, you can use functions like `CFStringCreateWithPascalString` to do so.

Version Notes

Carbon Porting Notes

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

Deprecated in Mac OS X v10.4.

Declared In
`PLStringFuncs.h`

Data Types

PEF2ContainerHeader

```
struct PEF2ContainerHeader {
    OSType tag1;
    OSType tag2;
    UInt32 currentFormat;
    UInt32 oldestFormat;
    UInt32 containerHeaderSize;
    UInt32 containerLength;
    UInt32 checksum;
    UInt32 sectionHeadersOffset;
    UInt32 sectionHeaderSize;
    UInt32 totalSectionCount;
    UInt32 instSectionCount;
    UInt32 loaderSectionIndex;
    UInt32 containerStringsOffset;
    UInt32 containerStringsLength;
    UInt32 options;
    UInt32 preferredAddress;
    UInt8 alignment;
    UInt8 stringEncoding;
    UInt16 reservedA;
    UInt32 reservedB;
    UInt32 reservedC;
    UInt32 nameOffset;
    OSType architecture;
    UInt32 dateTimeStamp;
    UInt32 currentVersion;
    UInt32 oldDefVersion;
    UInt32 oldImpVersion;
    UInt32 reservedD;
    UInt32 reservedE;
};
typedef struct PEF2ContainerHeader PEF2ContainerHeader;
```

Fields

`tag1`

`tag2`

`currentFormat`

`oldestFormat`

`containerHeaderSize`

`containerLength`

`checksum`

`sectionHeadersOffset`

`sectionHeaderSize`

`totalSectionCount`

`instSectionCount`

`loaderSectionIndex`

`containerStringsOffset`

`containerStringsLength`

```
options
preferredAddress
alignment
stringEncoding
reservedA
reservedB
reservedC
nameOffset
architecture
dateTimeStamp
currentVersion
oldDefVersion
oldImpVersion
reservedD
reservedE
```

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
`PEFBinaryFormat.h`

PEF2ExportedSymbolKey

```
typedef PEFExportedSymbolKey PEF2ExportedSymbolKey;
```

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
`PEFBinaryFormat.h`

PEF2ImportedLibrary

```
struct PEF2ImportedLibrary {
    UInt32 nameOffset;
    UInt32 oldImpVersion;
    UInt32 currentVersion;
    UInt32 importedSymbolCount;
    UInt32 firstImportedSymbol;
    UInt32 options;
    UInt32 reservedA;
};
typedef struct PEF2ImportedLibrary PEF2ImportedLibrary;
```

Fields

`nameOffset`

`oldImpVersion`

`currentVersion`

`importedSymbolCount`

`firstImportedSymbol`

`options`

`reservedA`

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
`PEFBinaryFormat.h`

PEF2LgExportedSymbolHashSlot

```
struct PEF2LgExportedSymbolHashSlot {
    UInt32 chainCount;
    UInt32 chainOffset;
};
typedef struct PEF2LgExportedSymbolHashSlot PEF2LgExportedSymbolHashSlot;
```

Fields

`chainCount`

`chainOffset`

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
`PEFBinaryFormat.h`

PEF2LgExportedSymbol

```
struct PEF2LgExportedSymbol {
    UInt8 symClass;
    UInt8 flags;
    UInt16 reservedA;
    UInt32 nameOffset;
    UInt32 versionPair;
    SInt32 sectionIndex;
    UInt32 sectionOffset;
    UInt32 reservedB;
};
typedef struct PEF2LgExportedSymbol PEF2LgExportedSymbol;
```

Fields

`symClass`

`flags`

`reservedA`

`nameOffset`

`versionPair`

`sectionIndex`

`sectionOffset`

`reservedB`

Discussion

Version Notes

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Declared In

`PEFBinaryFormat.h`

PEF2LgImportedSymbol

```
struct PEF2LgImportedSymbol {
    UInt8 symClass;
    UInt8 flags;
    UInt16 reservedA;
    UInt32 nameOffset;
    UInt32 versionPair;
    UInt32 reservedB;
};
typedef struct PEF2LgImportedSymbol PEF2LgImportedSymbol;
```

Fields

`symClass`

`flags`

`reservedA`

`nameOffset`

`versionPair`

`reservedB`

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
`PEFBinaryFormat.h`

PEF2LoaderInfoHeader

```
struct PEF2LoaderInfoHeader {
    UInt32 headerSize;
    UInt32 options;
    SInt32 mainSection;
    UInt32 mainOffset;
    SInt32 initSection;
    UInt32 initOffset;
    SInt32 termSection;
    UInt32 termOffset;
    SInt32 notifySection;
    UInt32 notifyOffset;
    UInt32 importedLibrariesOffset;
    UInt32 importedLibrarySize;
    UInt32 importedLibraryCount;
    UInt32 importedSymbolsOffset;
    UInt32 importedSymbolSize;
    UInt32 totalImportedSymbolCount;
    UInt32 loaderNamesOffset;
    UInt32 loaderNamesLength;
    UInt32 exportHashTableOffset;
    UInt8 exportHashTablePower;
    UInt8 reservedA;
    UInt16 reservedB;
    UInt32 exportedKeysOffset;
    UInt32 exportedSymbolsOffset;
    UInt32 exportedSymbolSize;
    UInt32 exportedSymbolCount;
    UInt32 relocHeadersOffset;
    UInt32 relocHeaderCount;
    UInt32 relocInstrOffset;
    UInt32 relocInstrLength;
    UInt32 reservedC;
    UInt32 reservedD;
};
typedef struct PEF2LoaderInfoHeader PEF2LoaderInfoHeader;
```

Fields

`headerSize`
`options`
`mainSection`
`mainOffset`
`initSection`
`initOffset`
`termSection`
`termOffset`
`notifySection`
`notifyOffset`
`importedLibrariesOffset`
`importedLibrarySize`
`importedLibraryCount`
`importedSymbolsOffset`
`importedSymbolSize`
`totalImportedSymbolCount`

```
loaderNamesOffset
loaderNamesLength
exportHashTableOffset
exportHashTablePower
reservedA
reservedB
exportedKeysOffset
exportedSymbolsOffset
exportedSymbolSize
exportedSymbolCount
relocHeadersOffset
relocHeaderCount
relocInstrOffset
relocInstrLength
reservedC
reservedD
```

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
PEFBinaryFormat.h

PEF2LoaderRelocationHeader

```
struct PEF2LoaderRelocationHeader {
    UInt32 sectionIndex;
    UInt32 relocLength;
    UInt32 firstRelocOffset;
    UInt32 reservedA;
};
typedef struct PEF2LoaderRelocationHeader PEF2LoaderRelocationHeader;
```

Fields

```
sectionIndex
relocLength
firstRelocOffset
reservedA
```

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
PEFBinaryFormat.h

PEF2SectionHeader

```
struct PEF2SectionHeader {
    UInt32 nameOffset;
    UInt32 presumedAddress;
    UInt32 totalLength;
    UInt32 unpackedLength;
    UInt32 containerLength;
    UInt32 containerOffset;
    UInt32 options;
    UInt8 shareKind;
    UInt8 alignment;
    UInt16 reservedA;
    UInt32 reservedB;
    UInt32 reservedC;
};
typedef struct PEF2SectionHeader PEF2SectionHeader;
```

Fields

`nameOffset`
`presumedAddress`
`totalLength`
`unpackedLength`
`containerLength`
`containerOffset`
`options`
`shareKind`
`alignment`
`reservedA`
`reservedB`
`reservedC`

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
`PEFBinaryFormat.h`

PEF2SmExportedSymbolHashSlot

```
typedef PEFExportedSymbolHashSlot PEF2SmExportedSymbolHashSlot;
```

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
PEFBinaryFormat.h

PEF2SmExportedSymbol

```
typedef PEFExportedSymbol PEF2SmExportedSymbol;
```

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
PEFBinaryFormat.h

PEF2SmImportedSymbol

```
typedef PEFImportedSymbol PEF2SmImportedSymbol;
```

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
PEFBinaryFormat.h

PEFContainerHeader

```
struct PEFContainerHeader {
    OSType tag1;
    OSType tag2;
    OSType architecture;
    UInt32 formatVersion;
    UInt32 dateTimeStamp;
    UInt32 oldDefVersion;
    UInt32 oldImpVersion;
    UInt32 currentVersion;
    UInt16 sectionCount;
    UInt16 instSectionCount;
    UInt32 reservedA;
};
typedef struct PEFContainerHeader PEFContainerHeader;
```

Fields

tag1

tag2

architecture

formatVersion

dateTimeStamp

oldDefVersion

oldImpVersion

currentVersion

sectionCount

instSectionCount

reservedA

Discussion

Version Notes

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Declared In

PEFBinaryFormat.h

PEFExportedSymbol

```
struct PEFExportedSymbol {
    UInt32 classAndName;
    UInt32 symbolValue;
    SInt16 sectionIndex;
};
typedef struct PEFExportedSymbol PEFExportedSymbol;
typedef PEFExportedSymbol PEF2SmExportedSymbol;
```

Fields

`classAndName`

`symbolValue`

`sectionIndex`

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
`PEFBinaryFormat.h`

PEFExportedSymbolHashSlot

```
struct PEFExportedSymbolHashSlot {
    UInt32 countAndStart;
};
typedef struct PEFExportedSymbolHashSlot PEFExportedSymbolHashSlot;
typedef PEFExportedSymbolHashSlot XLibExportedSymbolHashSlot;
```

Fields

`countAndStart`

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
`PEFBinaryFormat.h`

PEFExportedSymbolKey

```
struct PEFExportedSymbolKey {
    union {
        UInt32 fullHashWord;
        PEFSplitHashWord splitHashWord;
    } u;
};
typedef struct PEFExportedSymbolKey PEFExportedSymbolKey;
typedef PEFExportedSymbolKey XLibExportedSymbolKey;
```

Fields

`fullHashWord`

`splitHashWord`

Discussion

Version Notes

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Declared In

`PEFBinaryFormat.h`

PEFImportedLibrary

```
struct PEFImportedLibrary {
    UInt32 nameOffset;
    UInt32 oldImpVersion;
    UInt32 currentVersion;
    UInt32 importedSymbolCount;
    UInt32 firstImportedSymbol;
    UInt8 options;
    UInt8 reservedA;
    UInt16 reservedB;
};
typedef struct PEFImportedLibrary PEFImportedLibrary;
```

Fields

`nameOffset`

`oldImpVersion`

`currentVersion`

`importedSymbolCount`

`firstImportedSymbol`

`options`

`reservedA`

`reservedB`

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
`PEFBinaryFormat.h`

PEFImportedSymbol

```
struct PEFImportedSymbol {
    UInt32 classAndName;
};
typedef struct PEFImportedSymbol PEFImportedSymbol;
typedef PEFImportedSymbol PEF2SmImportedSymbol;
```

Fields

`classAndName`

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
`PEFBinaryFormat.h`

PEFLoaderInfoHeader

```
struct PEFLoaderInfoHeader {
    SInt32 mainSection;
    UInt32 mainOffset;
    SInt32 initSection;
    UInt32 initOffset;
    SInt32 termSection;
    UInt32 termOffset;
    UInt32 importedLibraryCount;
    UInt32 totalImportedSymbolCount;
    UInt32 relocSectionCount;
    UInt32 relocInstrOffset;
    UInt32 loaderStringsOffset;
    UInt32 exportHashOffset;
    UInt32 exportHashTablePower;
    UInt32 exportedSymbolCount;
};
typedef struct PEFLoaderInfoHeader PEFLoaderInfoHeader;
```

Fields

`mainSection`
`mainOffset`
`initSection`
`initOffset`
`termSection`
`termOffset`
`importedLibraryCount`
`totalImportedSymbolCount`
`relocSectionCount`
`relocInstrOffset`
`loaderStringsOffset`
`exportHashOffset`
`exportHashTablePower`
`exportedSymbolCount`

Discussion

Version Notes

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Declared In

`PEFBinaryFormat.h`

PEFLoaderRelocationHeader

```
struct PEFLoaderRelocationHeader {
    UInt16 sectionIndex;
    UInt16 reservedA;
    UInt32 relocCount;
    UInt32 firstRelocOffset;
};
typedef struct PEFLoaderRelocationHeader PEFLoaderRelocationHeader;
```

Fields

`sectionIndex`

`reservedA`

`relocCount`

`firstRelocOffset`

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
`PEFBinaryFormat.h`

PEFRelocChunk

```
typedef UInt16 PEFRelocChunk;
```

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
`PEFBinaryFormat.h`

PEFSectionHeader

```
struct PEFSectionHeader {
    SInt32 nameOffset;
    UInt32 defaultAddress;
    UInt32 totalLength;
    UInt32 unpackedLength;
    UInt32 containerLength;
    UInt32 containerOffset;
    UInt8 sectionKind;
    UInt8 shareKind;
    UInt8 alignment;
    UInt8 reservedA;
};
typedef struct PEFSectionHeader PEFSectionHeader;
```

Fields

`nameOffset`

`defaultAddress`

`totalLength`

`unpackedLength`

`containerLength`

`containerOffset`

`sectionKind`

`shareKind`

`alignment`

`reservedA`

Discussion

Version Notes

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Declared In

`PEFBinaryFormat.h`

PEFSplitHashWord

```
struct PEFSplitHashWord {
    UInt16 nameLength;
    UInt16 hashValue;
};
typedef struct PEFSplitHashWord PEFSplitHashWord;
```

Fields

`nameLength`

`hashValue`

Discussion

Version Notes

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Declared In

`PEFBinaryFormat.h`

XLibContainerHeader

```
struct XLibContainerHeader {
    OSType tag1;
    OSType tag2;
    UInt32 currentFormat;
    UInt32 containerStringsOffset;
    UInt32 exportHashOffset;
    UInt32 exportKeyOffset;
    UInt32 exportSymbolOffset;
    UInt32 exportNamesOffset;
    UInt32 exportHashTablePower;
    UInt32 exportedSymbolCount;
    UInt32 fragNameOffset;
    UInt32 fragNameLength;
    UInt32 dylibPathOffset;
    UInt32 dylibPathLength;
    OSType cpuFamily;
    OSType cpuModel;
    UInt32 dateTimeStamp;
    UInt32 currentVersion;
    UInt32 oldDefVersion;
    UInt32 oldImpVersion;
};
typedef struct XLibContainerHeader XLibContainerHeader;
```

Fields

tag1

tag2

currentFormat

containerStringsOffset

exportHashOffset

exportKeyOffset

exportSymbolOffset

exportNamesOffset

exportHashTablePower

exportedSymbolCount

fragNameOffset

fragNameLength

dylibPathOffset

dylibPathLength

cpuFamily

cpuModel

dateTimeStamp

currentVersion

oldDefVersion

oldImpVersion

Discussion

Version Notes

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Declared In
PEFBinaryFormat.h

XLibExportedSymbol

```
struct XLibExportedSymbol {
    UInt32 classAndName;
    UInt32 bpOffset;
};
typedef struct XLibExportedSymbol XLibExportedSymbol;
```

Fields
classAndName
bpOffset

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
PEFBinaryFormat.h

XLibExportedSymbolHashSlot

```
typedef PEFExportedSymbolHashSlot XLibExportedSymbolHashSlot;
```

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
PEFBinaryFormat.h

XLibExportedSymbolKey

```
typedef PEFExportedSymbolKey XLibExportedSymbolKey;
```

Discussion

Version Notes

Carbon Porting Notes

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

Declared In
`PEFBinaryFormat.h`

Constants

kPEF2IsReexportLibraryMask

```
enum {
    kPEF2IsReexportLibraryMask = 0x00000001,
    kPEF2IsGlueLibraryMask = 0x00000002
};
```

Constants

`kPEF2IsReexportLibraryMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2IsGlueLibraryMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEF2LdrInfoLargeImpSymMask

```
enum {
    kPEF2LdrInfoLargeImpSymMask = 0x00000001,
    kPEF2LdrInfoLargeExpSymMask = 0x00000002,
    kPEF2LdrInfoLargeExpHashMask = 0x00000004
};
```

Constants

`kPEF2LdrInfoLargeImpSymMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2LdrInfoLargeExpSymMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2LdrInfoLargeExpHashMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEF2PrivateShare

```
enum {
    kPEF2PrivateShare = 0,
    kPEF2ProcessShare = 1,
    kPEF2GlobalShare = 4,
    kPEF2ProtectedShare = 5
};
```

Constants

`kPEF2PrivateShare`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2ProcessShare`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2GlobalShare`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2ProtectedShare`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEF2SectionHasCodeMask

```
enum {
    kPEF2SectionHasCodeMask = 0x00000001,
    kPEF2SectionIsWriteableMask = 0x00000002,
    kPEF2SectionHasRelocationsMask = 0x00000004,
    kPEF2SectionContentsArePackedMask = 0x00000100,
    kPEF2SectionNoZeroFillMask = 0x00000200,
    kPEF2SectionResidentMask = 0x00000400,
    kPEF2SectionFollowsPriorMask = 0x00010000,
    kPEF2SectionPrecedesNextMask = 0x00020000,
    kPEF2SectionHasLoaderTablesMask = 0x01000000,
    kPEF2SectionHasDebugTablesMask = 0x02000000,
    kPEF2SectionHasExceptionTablesMask = 0x04000000,
    kPEF2SectionHasTracebackTablesMask = 0x08000000
};
```

Constants

`kPEF2SectionHasCodeMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2SectionIsWriteableMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2SectionHasRelocationsMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2SectionContentsArePackedMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2SectionNoZeroFillMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2SectionResidentMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2SectionFollowsPriorMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2SectionPrecedesNextMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2SectionHasLoaderTablesMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2SectionHasDebugTablesMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2SectionHasExceptionTablesMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2SectionHasTracebackTablesMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEF2StringsAreASCII

```
enum {
    kPEF2StringsAreASCII = 0,
    kPEF2StringsAreUnicode = 1
};
```

Constants

`kPEF2StringsAreASCII`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2StringsAreUnicode`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEF2Tag1

```
enum {
    kPEF2Tag1 = kPEFTag1,
    kPEF2Tag2 = 'PEF ',
    kPEF2CurrentFormat = 0x00000002,
    kPEF2OldestHandler = 0x00000002
};
```

Constants

`kPEF2Tag1`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2Tag2`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2CurrentFormat`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2OldestHandler`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEF2WeakImportLibMask

```
enum {
    kPEF2WeakImportLibMask = kPEFWeakImportLibMask,
    kPEF2InitLibBeforeMask = kPEFInitLibBeforeMask
};
```

Constants

`kPEF2WeakImportLibMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEF2InitLibBeforeMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFAbsoluteExport

```
enum {
    kPEFAbsoluteExport = -2,
    kPEFReexportedImport = -3
};
```

Constants

`kPEFAbsoluteExport`

Available in Mac OS X v10.0 and later.

Declared in `PEFBinaryFormat.h`.

`kPEFReexportedImport`

Available in Mac OS X v10.0 and later.

Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFCodeSection

```
enum {
    kPEFCodeSection = 0,
    kPEFUnpackedDataSection = 1,
    kPEFPackedDataSection = 2,
    kPEFConstantSection = 3,
    kPEFExecDataSection = 6,
    kPEFLoaderSection = 4,
    kPEFDebugSection = 5,
    kPEFExceptionSection = 7,
    kPEFTracebackSection = 8
};
```

Constants

`kPEFCodeSection`

Available in Mac OS X v10.0 and later.

Declared in `PEFBinaryFormat.h`.

`kPEFUnpackedDataSection`

Available in Mac OS X v10.0 and later.

Declared in `PEFBinaryFormat.h`.

`kPEFPackedDataSection`

Available in Mac OS X v10.0 and later.

Declared in `PEFBinaryFormat.h`.

`kPEFConstantSection`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFExecDataSection`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFLoaderSection`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFDebugSection`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFExceptionSection`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFTracebackSection`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFCodeSymbol

```
enum {
    kPEFCodeSymbol = 0x00,
    kPEFDataSymbol = 0x01,
    kPEFTVectorSymbol = 0x02,
    kPEFTOCSymbol = 0x03,
    kPEFGlueSymbol = 0x04,
    kPEFUndefinedSymbol = 0x0F,
    kPEFWeakImportSymMask = 0x80
};
```

Constants

`kPEFCodeSymbol`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFDataSymbol`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFTVectorSymbol`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFTOCSymbol`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFGlueSymbol`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFUndefinedSymbol`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFWeakImportSymMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFExpSymClassShift

```
enum {
    kPEFExpSymClassShift = 24,
    kPEFExpSymNameOffsetMask = 0x00FFFFFF,
    kPEFExpSymMaxNameOffset = 0x00FFFFFF
};
```

Constants

`kPEFExpSymClassShift`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFExpSymNameOffsetMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFExpSymMaxNameOffset`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFFirstSectionHeaderOffset

```
enum {
    kPEFFirstSectionHeaderOffset = sizeof(PEFContainerHeader)
};
```

Constants

`kPEFFirstSectionHeaderOffset`

Available in Mac OS X v10.0 and later.

Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFHashLengthShift

```
enum {
    kPEFHashLengthShift = 16,
    kPEFHashValueMask = 0x0000FFFF,
    kPEFHashMaxLength = 0x0000FFFF
};
```

Constants

`kPEFHashLengthShift`

Available in Mac OS X v10.0 and later.

Declared in `PEFBinaryFormat.h`.

`kPEFHashValueMask`

Available in Mac OS X v10.0 and later.

Declared in `PEFBinaryFormat.h`.

`kPEFHashMaxLength`

Available in Mac OS X v10.0 and later.

Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFHashSlotSymCountShift

```
enum {
    kPEFHashSlotSymCountShift = 18,
    kPEFHashSlotFirstKeyMask = 0x0003FFFF,
    kPEFHashSlotMaxSymbolCount = 0x00003FFF,
    kPEFHashSlotMaxKeyIndex = 0x0003FFFF
};
```

Constants

`kPEFHashSlotSymCountShift`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFHashSlotFirstKeyMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFHashSlotMaxSymbolCount`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFHashSlotMaxKeyIndex`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFImpSymClassShift

```
enum {
    kPEFImpSymClassShift = 24,
    kPEFImpSymNameOffsetMask = 0x00FFFFFF,
    kPEFImpSymMaxNameOffset = 0x00FFFFFF
};
```

Constants

`kPEFImpSymClassShift`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFImpSymNameOffsetMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFImpSymMaxNameOffset`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFPkDataOpcodeShift

```
enum {
    kPEFPkDataOpcodeShift = 5,
    kPEFPkDataCount5Mask = 0x1F,
    kPEFPkDataMaxCount5 = 31,
    kPEFPkDataVCountShift = 7,
    kPEFPkDataVCountMask = 0x7F,
    kPEFPkDataVCountEndMask = 0x80
};
```

Constants

`kPEFPkDataOpcodeShift`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFPkDataCount5Mask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFPkDataMaxCount5`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFPkDataVCountShift`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFPkDataVCountMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFPkDataVCountEndMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFPkDataZero

```
enum {
    kPEFPkDataZero = 0,
    kPEFPkDataBlock = 1,
    kPEFPkDataRepeat = 2,
    kPEFPkDataRepeatBlock = 3,
    kPEFPkDataRepeatZero = 4
};
```

Constants

`kPEFPkDataZero`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFPkDataBlock`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFPkDataRepeat`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFPkDataRepeatBlock`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFPkDataRepeatZero`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFProcessShare

```
enum {
    kPEFProcessShare = 1,
    kPEFGlobalShare = 4,
    kPEFProtectedShare = 5
};
```

Constants

`kPEFProcessShare`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFGlobalShare`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFProtectedShare`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFRelocBasicOpcodeRange

```
enum {
    kPEFRelocBasicOpcodeRange = 128
};
```

Constants

`kPEFRelocBasicOpcodeRange`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFRelocBySectDWithSkip

```
enum {
    kPEFRelocBySectDWithSkip = 0x00,
    kPEFRelocBySectC = 0x20,
    kPEFRelocBySectD = 0x21,
    kPEFRelocTVector12 = 0x22,
    kPEFRelocTVector8 = 0x23,
    kPEFRelocVTable8 = 0x24,
    kPEFRelocImportRun = 0x25,
    kPEFRelocSmByImport = 0x30,
    kPEFRelocSmSetSectC = 0x31,
    kPEFRelocSmSetSectD = 0x32,
    kPEFRelocSmBySection = 0x33,
    kPEFRelocIncrPosition = 0x40,
    kPEFRelocSmRepeat = 0x48,
    kPEFRelocSetPosition = 0x50,
    kPEFRelocLgByImport = 0x52,
    kPEFRelocLgRepeat = 0x58,
    kPEFRelocLgSetOrBySection = 0x5A,
    kPEFRelocUndefinedOpcode = 0xFF
};
```

Constants

`kPEFRelocBySectDWithSkip`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocBySectC`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocBySectD`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocTVector12`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocTVector8`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocVTable8`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocImportRun`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocSmByImport`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocSmSetSectC`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocSmSetSectD`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocSmBySection`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocIncrPosition`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocSmRepeat`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocSetPosition`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocLgByImport`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocLgRepeat`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocLgSetOrBySection`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocUndefinedOpcode`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFRelocIncrPositionMaxOffset

```
enum {
    kPEFRelocIncrPositionMaxOffset = 4096
};
```

Constants

`kPEFRelocIncrPositionMaxOffset`

Available in Mac OS X v10.0 and later.

Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFRelocLgByImportMaxIndex

```
enum {
    kPEFRelocLgByImportMaxIndex = 0x03FFFFFF
};
```

Constants

`kPEFRelocLgByImportMaxIndex`

Available in Mac OS X v10.0 and later.

Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFRelocLgBySectionSubopcode

```
enum {
    kPEFRelocLgBySectionSubopcode = 0x00,
    kPEFRelocLgSetSectCSubopcode = 0x01,
    kPEFRelocLgSetSectDSubopcode = 0x02
};
```

Constants

`kPEFRelocLgBySectionSubopcode`

Available in Mac OS X v10.0 and later.

Declared in `PEFBinaryFormat.h`.

`kPEFRelocLgSetSectCSubopcode`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocLgSetSectDSubopcode`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFRelocLgRepeatMaxChunkCount

```
enum {
    kPEFRelocLgRepeatMaxChunkCount = 16,
    kPEFRelocLgRepeatMaxRepeatCount = 0x003FFFFF
};
```

Constants

`kPEFRelocLgRepeatMaxChunkCount`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocLgRepeatMaxRepeatCount`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFRelocLgSetOrBySectionMaxIndex

```
enum {
    kPEFRelocLgSetOrBySectionMaxIndex = 0x003FFFFF
};
```

Constants

`kPEFRelocLgSetOrBySectionMaxIndex`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFRelocRunMaxRunLength

```
enum {
    kPEFRelocRunMaxRunLength = 512
};
```

Constants

`kPEFRelocRunMaxRunLength`

Available in Mac OS X v10.0 and later.

Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFRelocSetPosMaxOffset

```
enum {
    kPEFRelocSetPosMaxOffset = 0x03FFFFFF
};
```

Constants

`kPEFRelocSetPosMaxOffset`

Available in Mac OS X v10.0 and later.

Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFRelocSmIndexMaxIndex

```
enum {
    kPEFRelocSmIndexMaxIndex = 511
};
```

Constants

`kPEFRelocSmIndexMaxIndex`

Available in Mac OS X v10.0 and later.

Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFRelocSmRepeatMaxChunkCount

```
enum {
    kPEFRelocSmRepeatMaxChunkCount = 16,
    kPEFRelocSmRepeatMaxRepeatCount = 256
};
```

Constants

`kPEFRelocSmRepeatMaxChunkCount`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocSmRepeatMaxRepeatCount`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFRelocWithSkipMaxSkipCount

```
enum {
    kPEFRelocWithSkipMaxSkipCount = 255,
    kPEFRelocWithSkipMaxRelocCount = 63
};
```

Constants

`kPEFRelocWithSkipMaxSkipCount`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFRelocWithSkipMaxRelocCount`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFTag1

```
enum {
    kPEFTag1 = 'Joy!',
    kPEFTag2 = 'peff',
    kPEFVersion = 0x00000001
};
```

Constants

`kPEFTag1`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFTag2`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFVersion`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kPEFWeakImportLibMask

```
enum {
    kPEFWeakImportLibMask = 0x40,
    kPEFInitLibBeforeMask = 0x80
};
```

Constants

`kPEFWeakImportLibMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kPEFInitLibBeforeMask`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

kXLibTag1

```
enum {
    kXLibTag1 = 'Mac',
    kVLibTag2 = 'VLib',
    kBLibTag2 = 'BLib',
    kXLibVersion = 0x00000001
};
```

Constants

`kXLibTag1`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kVLibTag2`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kBLibTag2`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

`kXLibVersion`
Available in Mac OS X v10.0 and later.
Declared in `PEFBinaryFormat.h`.

Discussion

Version Notes

Carbon Porting Notes

Power Manager Reference

Framework: CoreServices/CoreServices.h

Declared in Power.h

Overview

The Power Manager controls power to the internal hardware devices of battery-powered Macintosh computers (such PowerBook computers). The Power Manager automatically shuts off power to internal devices to conserve power whenever the computer has not been used for a predetermined amount of time. In addition, the Power Manager allows your application or other software to

- install a procedure that is executed when power to internal devices is about to be shut off or when power has just been restored
- set a timer to wake up the computer at some time in the future
- set or disable the wakeup timer and read its current setting
- enable, disable, or delay the CPU idle feature
- read the current CPU clock speed
- control power to the internal modem and serial ports
- read the status of the internal modem
- read the state of the battery charge and the status of the battery charger

Most applications do not need to know whether they are executing on a battery-powered Macintosh computer because the transition between power states is largely invisible. As a result, most applications do not need to use Power Manager routines. You need Power Manager only if you are writing a program--such as a device driver--that must control power to some subsystem of a battery-powered Macintosh computer or that might be affected by the idle or sleep state.

Carbon supports Power Manager functions prior to Power Manager 2.0. However, many of these functions do nothing on Mac OS X; these calls have been retained in Carbon as the only means for implementing power management on Mac OS 8 and 9. Before using any of the Power Manager API, you should call the `PMFeatures` function to check the availability of the feature you wish to use. On Mac OS X, use the functions provided in IOKit for power management. For more information on IOKit, see I/O Kit Fundamentals.

Functions by Task

Function descriptions are grouped by the tasks for which you use the functions. For an alphabetical list of functions, go to the API index at the end of the document.

Controlling the Idle State

`CurrentProcessorSpeed` (page 1593)
Returns the current effective clock speed of the CPU in megahertz.

`DisableIdle` (page 1595) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`EnableIdle` (page 1596) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`IdleUpdate` (page 1606) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use `UpdateSystemActivity` instead.)

Controlling and Reading the Wakeup Timer

`DisableWUTime` (page 1595) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`GetWUTime` (page 1604) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`SetWUTime` (page 1618) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

Controlling the Sleep Queue

`SleepQInstall` (page 1619)
Adds an entry to the sleep queue.

`SleepQRemove` (page 1619)
Removes an entry from the sleep queue.

Controlling Serial Power

`AOff` (page 1590) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`AOn` (page 1590) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`AOnIgnoreModem` (page 1590) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`BOff` (page 1592) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

BOn (page 1593) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

Reading the Status of the Internal Modem

ModemStatus (page 1610) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

Reading the Status of the Battery and of the Battery Charger

BatteryStatus (page 1592) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

Miscellaneous

BatteryCount (page 1591)

DisposeHDSpindownUPP (page 1596)
Unimplemented.
DisposePMgrStateChangeUPP (page 1596)
Unimplemented
DisposeSleepQUPP (page 1596)

GetCPUSpeed (page 1598)
Returns the current effective clock speed of the CPU in megahertz.
InvokeHDSpindownUPP (page 1607)
Unimplemented.
InvokePMgrStateChangeUPP (page 1607)
Unimplemented.
InvokeSleepQUPP (page 1608)

MaximumProcessorSpeed (page 1610)

MinimumProcessorSpeed (page 1610)

NewHDSpindownUPP (page 1611)
Unimplemented.
NewPMgrStateChangeUPP (page 1611)
Unimplemented.
NewSleepQUPP (page 1612)

UpdateSystemActivity (page 1620)

`FullProcessorSpeed` (page 1597) Deprecated in Mac OS X v10.5
Unimplemented.

`PMFeatures` (page 1612) Deprecated in Mac OS X v10.5

`PMSelectorCount` (page 1613) Deprecated in Mac OS X v10.5
Unimplemented.

`SetProcessorSpeed` (page 1615) Deprecated in Mac OS X v10.5
Unimplemented.

`SetSpindownDisable` (page 1617) Deprecated in Mac OS X v10.5
Unimplemented.

`AutoSleepControl` (page 1591) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`DelaySystemIdle` (page 1594) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`DimmingControl` (page 1594) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`EnableProcessorCycling` (page 1597) Deprecated in Mac OS X v10.0
Unimplemented.

`GetBatteryTimes` (page 1598) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`GetBatteryVoltage` (page 1598) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`GetDimmingTimeout` (page 1599) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`GetDimSuspendState` (page 1599) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`GetHardDiskTimeout` (page 1600) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`GetIntModemInfo` (page 1600) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`GetLastActivity` (page 1601) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`GetScaledBatteryInfo` (page 1601) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`GetSCSIDiskModeAddress` (page 1602) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`GetSleepTimeout` (page 1602) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`GetSoundMixerState` (page 1603) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`GetStartupTimer` (page 1603) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

`GetWakeupTimer` (page 1604) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

HardDiskPowered (page 1605) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

HardDiskQInstall (page 1605) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

HardDiskQRemove (page 1606) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

IsAutoSlpControlDisabled (page 1608) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

IsDimmingControlDisabled (page 1608) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

IsProcessorCyclingEnabled (page 1609) Deprecated in Mac OS X v10.0
Unimplemented.

IsSpindownDisabled (page 1609) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

PMgrStateQInstall (page 1612) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

PMgrStateQRemove (page 1613) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

SetDimmingTimeout (page 1614) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

SetDimSuspendState (page 1614) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

SetHardDiskTimeout (page 1615) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

SetIntModemState (page 1615) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

SetSCSIDiskModeAddress (page 1616) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

SetSleepTimeout (page 1616) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

SetSoundMixerState (page 1617) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

SetStartupTimer (page 1617) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

SetWakeupTimer (page 1618) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

SpinDownHardDisk (page 1620) Deprecated in Mac OS X v10.0
Unimplemented. (Deprecated. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

Functions

AOff

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void AOff (
   void
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

AOn

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void AOn (
   void
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

AOnIgnoreModem

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void AOnIgnoreModem (
   void
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

AutoSleepControl

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void AutoSleepControl (
   Boolean enableSleep
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

BatteryCount

```
short BatteryCount (
   void
);
```

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

Declared In
`Power.h`

BatteryStatus

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
OSErr BatteryStatus (
   Byte *status,
   Byte *power
);
```

Parameters

Status

On return, the referenced value holds the status of the battery charger and the voltage level of the battery, described in BatteryByte Bits (page 1631).The connChangedBit is set when the charger connection is changed—either connected or disconnected. When this bit is set, the Power Manager IC sends an interrupt to the CPU.

The `batteryLowBit` is set whenever battery voltage drops below the value set in parameter RAM. The Power Manager IC sends an interrupt to the CPU once every second when battery voltage is low.If the `batteryDeadBit` were set, it would indicate a dead battery; however, the Power Manager automatically shuts the system down when the battery voltage drops below a preset level, so this bit is always 0.

Power

On return, the referenced value contains the `Power` value you can use to estimate the battery voltage:

voltage = ((`Power`/100) + 5.12) volts

Due to the nature of lead-acid batteries, the battery power remaining is difficult to measure accurately. Temperature, load, and other factors can alter the measured voltage by 30 percent or more. The Power Manager takes as many of these factors into account as possible, but the voltage measurement can still be in error by up to 10 percent. The measurement is most accurate when the Macintosh Portable has been in the sleep state for at least 30 minutes.

Return Value

A result code. See "Power Manager Result Codes" (page 1654).

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

BOff

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void BOff (
   void
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

BOn

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void BOn (
   void
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

CurrentProcessorSpeed

Returns the current effective clock speed of the CPU in megahertz.

```
short CurrentProcessorSpeed (
   void
);
```

Return Value
The clock speed of the CPU in megahertz (MHz). One MHz represents one million cycles per second.

Special Considerations

Prior to Mac OS X 10.4, this function returns the maximum clock speed, not the current effective clock speed.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Power.h`

DelaySystemIdle

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
OSErr DelaySystemIdle (
   void
);
```

Return Value

A result code. See "Power Manager Result Codes" (page 1654).

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

DimmingControl

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void DimmingControl (
   Boolean enableSleep
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

DisableIdle

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void DisableIdle (
   void
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

DisableWUTime

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
OSErr DisableWUTime (
   void
);
```

Return Value
A result code. See "Power Manager Result Codes" (page 1654).

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

DisposeHDSpindownUPP

Unimplemented.

```
void DisposeHDSpindownUPP (
   HDSpindownUPP userUPP
);
```

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

Declared In
`Power.h`

DisposePMgrStateChangeUPP

Unimplemented

```
void DisposePMgrStateChangeUPP (
   PMgrStateChangeUPP userUPP
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

Declared In
`Power.h`

DisposeSleepQUPP

```
void DisposeSleepQUPP (
   SleepQUPP userUPP
);
```

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

Declared In
`Power.h`

EnableIdle

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void EnableIdle (
   void
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

EnableProcessorCycling

Unimplemented. (Deprecated in Mac OS X v10.0.)

```
void EnableProcessorCycling (
   Boolean enable
);
```

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
`Power.h`

FullProcessorSpeed

Unimplemented. (Deprecated in Mac OS X v10.5.)

```
Boolean FullProcessorSpeed (
   void
);
```

Return Value
See the Mac Types documentation for a description of the `Boolean` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Power.h`

GetBatteryTimes

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void GetBatteryTimes (
   short whichBattery,
   BatteryTimeRec *theTimes
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

GetBatteryVoltage

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
Fixed GetBatteryVoltage (
   short whichBattery
);
```

Return Value

See the Mac Types documentation for a description of the `Fixed` data type.

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

GetCPUSpeed

Returns the current effective clock speed of the CPU in megahertz.

```
long GetCPUSpeed (
   void
);
```

Return Value
The clock speed of the CPU in megahertz.

Discussion
For more information, see `CurrentProcessorSpeed` (page 1593).

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

Declared In
`Power.h`

GetDimmingTimeout

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
UInt8 GetDimmingTimeout (
   void
);
```

Return Value
See the Mac Types documentation for a description of the `UInt8` data type.

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

GetDimSuspendState

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
Boolean GetDimSuspendState (
   void
);
```

Return Value
See the Mac Types documentation for a description of the `Boolean` data type.

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

GetHardDiskTimeout

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
UInt8 GetHardDiskTimeout (
   void
);
```

Return Value

See the Mac Types documentation for a description of the `UInt8` data type.

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

GetIntModemInfo

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
UInt32 GetIntModemInfo (
   void
);
```

Return Value

See the Mac Types documentation for a description of the `UInt32` data type.

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

GetLastActivity

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
OSErr GetLastActivity (
   ActivityInfo *theActivity
);
```

Return Value

A result code. See "Power Manager Result Codes" (page 1654).

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

GetScaledBatteryInfo

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void GetScaledBatteryInfo (
   short whichBattery,
   BatteryInfo *theInfo
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

GetSCSIDiskModeAddress

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
short GetSCSIDiskModeAddress (
   void
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

GetSleepTimeout

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
UInt8 GetSleepTimeout (
   void
);
```

Return Value
See the Mac Types documentation for a description of the `UInt8` data type.

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

GetSoundMixerState

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
OSErr GetSoundMixerState (
   SoundMixerByte *theSoundMixerByte
);
```

Return Value
A result code. See "Power Manager Result Codes" (page 1654).

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

GetStartupTimer

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
OSErr GetStartupTimer (
   StartupTime *theTime
);
```

Return Value
A result code. See "Power Manager Result Codes" (page 1654).

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

GetWakeupTimer

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void GetWakeupTimer (
   WakeupTime *theTime
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

GetWUTime

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
OSErr GetWUTime (
   long *wuTime,
   Byte *wuFlag
);
```

Parameters

WUTime

On return, the referenced value holds the current setting of the wakeup timer specified as the number of seconds since midnight, January 1, 1904.

WUFlag

On return, the low order bit of the referenced value is set to 1 if and only if the wakeup timer is enabled. The other bits in `WUFlag` are reserved.

Return Value
A result code. See "Power Manager Result Codes" (page 1654).

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

HardDiskPowered

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
Boolean HardDiskPowered (
   void
);
```

Return Value
See the Mac Types documentation for a description of the `Boolean` data type.

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

HardDiskQInstall

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
OSErr HardDiskQInstall (
   HDQueueElement *theElement
);
```

Return Value
A result code. See "Power Manager Result Codes" (page 1654).

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

HardDiskQRemove

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
OSErr HardDiskQRemove (
   HDQueueElement *theElement
);
```

Return Value
A result code. See "Power Manager Result Codes" (page 1654).

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

IdleUpdate

Unimplemented. (Deprecated in Mac OS X v10.0. Use `UpdateSystemActivity` instead.)

```
long IdleUpdate (
   void
);
```

Return Value
The `IdleUpdate` function returns the value in the `Ticks` global variable at the time the function was called.

Special Considerations
This function is unimplemented on Mac OS X. Use `UpdateSystemActivity(IdleActivity)` instead.

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
`Power.h`

InvokeHDSpindownUPP

Unimplemented.

```
void InvokeHDSpindownUPP (
   HDQueueElement *theElement,
   HDSpindownUPP userUPP
);
```

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

Declared In
`Power.h`

InvokePMgrStateChangeUPP

Unimplemented.

```
void InvokePMgrStateChangeUPP (
   PMgrQueueElement *theElement,
   long stateBits,
   PMgrStateChangeUPP userUPP
);
```

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

Declared In
`Power.h`

InvokeSleepQUPP

```
long InvokeSleepQUPP (
   long message,
   SleepQRecPtr qRecPtr,
   SleepQUPP userUPP
);
```

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

Declared In
`Power.h`

IsAutoSlpControlDisabled

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
Boolean IsAutoSlpControlDisabled (
   void
);
```

Return Value
See the Mac Types documentation for a description of the `Boolean` data type.

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

IsDimmingControlDisabled

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
Boolean IsDimmingControlDisabled (
   void
);
```

Return Value
See the Mac Types documentation for a description of the `Boolean` data type.

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

IsProcessorCyclingEnabled

Unimplemented. (Deprecated in Mac OS X v10.0.)

```
Boolean IsProcessorCyclingEnabled (
   void
);
```

Return Value

See the Mac Types documentation for a description of the `Boolean` data type.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.0.

Not available to 64-bit applications.

Declared In

`Power.h`

IsSpindownDisabled

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
Boolean IsSpindownDisabled (
   void
);
```

Return Value

See the Mac Types documentation for a description of the `Boolean` data type.

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

MaximumProcessorSpeed

```
short MaximumProcessorSpeed (
   void
);
```

Version Notes
`MaximumProcessorSpeed` is unimplemented on versions of Mac OS X prior to Mac OS X 10.1.

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

Declared In
`Power.h`

MinimumProcessorSpeed

```
short MinimumProcessorSpeed (
   void
);
```

Special Considerations
`MinimumProcessorSpeed` is unimplemented on versions of Mac OS X prior to Mac OS X v10.1.

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

Declared In
`Power.h`

ModemStatus

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
OSErr ModemStatus (
   Byte *status
);
```

Parameters
Status
On return, the referenced variable has its bits set as indicated in ModemByte Bits (page 1639).

Return Value
A result code. See "Power Manager Result Codes" (page 1654).

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

NewHDSpindownUPP

Unimplemented.

```
HDSpindownUPP NewHDSpindownUPP (
   HDSpindownProcPtr userRoutine
);
```

Return Value

See the description of the `HDSpindownUPP` data type.

Discussion

See the callback `HDSpindownProcPtr` (page 1620) for more information.

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Power.h`

NewPMgrStateChangeUPP

Unimplemented.

```
PMgrStateChangeUPP NewPMgrStateChangeUPP (
   PMgrStateChangeProcPtr userRoutine
);
```

Return Value

See the description of the `PMgrStateChangeUPP` data type.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Power.h`

NewSleepQUPP

```
SleepQUPP NewSleepQUPP (
   SleepQProcPtr userRoutine
);
```

Return Value
See the description of the `SleepQUPP` data type.

Discussion
See the callback `SleepQProcPtr` (page 1622) for more information.

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

Declared In
`Power.h`

PMFeatures

(Deprecated in Mac OS X v10.5.)

```
UInt32 PMFeatures (
   void
);
```

Return Value
See the Mac Types documentation for a description of the `UInt32` data type.

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
`Power.h`

PMgrStateQInstall

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
OSErr PMgrStateQInstall (
   PMgrQueueElement *theElement
);
```

Return Value
A result code. See "Power Manager Result Codes" (page 1654).

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

PMgrStateQRemove

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
OSErr PMgrStateQRemove (
   PMgrQueueElement *theElement
);
```

Return Value

A result code. See "Power Manager Result Codes" (page 1654).

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

PMSelectorCount

Unimplemented. (Deprecated in Mac OS X v10.5.)

```
short PMSelectorCount (
   void
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

SetDimmingTimeout

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void SetDimmingTimeout (
   UInt8 timeout
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

SetDimSuspendState

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void SetDimSuspendState (
   Boolean dimSuspendState
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

SetHardDiskTimeout

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void SetHardDiskTimeout (
   UInt8 timeout
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

SetIntModemState

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void SetIntModemState (
   short theState
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

SetProcessorSpeed

Unimplemented. (Deprecated in Mac OS X v10.5.)

```
Boolean SetProcessorSpeed (
   Boolean fullSpeed
);
```

Return Value
See the Mac Types documentation for a description of the `Boolean` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Power.h`

SetSCSIDiskModeAddress

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void SetSCSIDiskModeAddress (
   short scsiAddress
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

SetSleepTimeout

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void SetSleepTimeout (
   UInt8 timeout
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

SetSoundMixerState

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
OSErr SetSoundMixerState (
   SoundMixerByte *theSoundMixerByte
);
```

Return Value
A result code. See "Power Manager Result Codes" (page 1654).

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

SetSpindownDisable

Unimplemented. (Deprecated in Mac OS X v10.5.)

```
void SetSpindownDisable (
   Boolean setDisable
);
```

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
`Power.h`

SetStartupTimer

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
OSErr SetStartupTimer (
   StartupTime *theTime
);
```

Return Value
A result code. See "Power Manager Result Codes" (page 1654).

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

SetWakeupTimer

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void SetWakeupTimer (
   WakeupTime *theTime
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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
`Power.h`

SetWUTime

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
OSErr SetWUTime (
   long wuTime
);
```

Return Value
A result code. See "Power Manager Result Codes" (page 1654).

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

SleepQInstall

Adds an entry to the sleep queue.

```
void SleepQInstall (
   SleepQRecPtr qRecPtr
);
```

Parameters

qRecPtr

A pointer to a sleep queue record that you must provide.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Power.h`

SleepQRemove

Removes an entry from the sleep queue.

```
void SleepQRemove (
   SleepQRecPtr qRecPtr
);
```

Parameters

qRecPtr

A pointer to a sleep queue record that you provided when you added your routine to the sleep queue.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Power.h`

SpinDownHardDisk

Unimplemented. (Deprecated in Mac OS X v10.0. Use I/O Kit instead; see *I/O Kit Fundamentals*.)

```
void SpinDownHardDisk (
   void
);
```

Special Considerations

The I/O Kit Framework header file IOPMLib.h (in *I/O Kit Framework Reference*) provides access to common power management facilites, such as initiating system sleep, getting current idle timer values, registering for sleep/wake notifications, and preventing system sleep. For additional information about power management for device drivers, see *I/O Kit Fundamentals* and *I/O Kit Device Driver Design Guidelines*.

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

`Power.h`

UpdateSystemActivity

```
OSErr UpdateSystemActivity (
   UInt8 activity
);
```

Return Value

A result code. See "Power Manager Result Codes" (page 1654).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Power.h`

Callbacks

HDSpindownProcPtr

```
typedef void (*HDSpindownProcPtr) (
    HDQueueElement * theElement
);
```

If you name your function `MyHDSpindownCallback`, you would declare it like this:

```
void MyHDSpindownCallback (
    HDQueueElement * theElement
);
```

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

Declared In
`Power.h`

PMgrStateChangeProcPtr

```
typedef void (*PMgrStateChangeProcPtr) (
    PMgrQueueElement * theElement,
    long stateBits
);
```

If you name your function `MyPMgrStateChangeCallback`, you would declare it like this:

```
void MyPMgrStateChangeCallback (
    PMgrQueueElement * theElement,
    long stateBits
);
```

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

Declared In
`Power.h`

PowerHandlerProcPtr

```
typedef OSStatus (*PowerHandlerProcPtr) (
    UInt32 message,
    void * param,
    UInt32 refCon,
    RegEntryID * regEntryID
);
```

If you name your function `MyPowerHandlerCallback`, you would declare it like this:

```
OSStatus MyPowerHandlerCallback (
    UInt32 message,
    void * param,
    UInt32 refCon,
    RegEntryID * regEntryID
);
```

Return Value
A result code. See "Power Manager Result Codes" (page 1654).

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

Declared In
`Power.h`

SleepQProcPtr

```
typedef long (*SleepQProcPtr) (
    long message,
    SleepQRecPtr qRecPtr
);
```

If you name your function `MySleepQProc`, you would declare it like this:

```
long MySleepQProc (
    long message,
    SleepQRecPtr qRecPtr
);
```

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

Declared In
`Power.h`

Data Types

ActivityInfo

```
struct ActivityInfo {
    short ActivityType;
    unsigned long ActivityTime;
};
typedef struct ActivityInfo ActivityInfo;
```

Fields

`ActivityType`
A short representing the type of activity to fetch.

`ActivityTime`
An unsigned long representing the time of the last activity in ticks.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Power.h`

BatteryByte

```
typedef SInt8 BatteryByte;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Power.h

BatteryInfo

```
struct BatteryInfo {
    UInt8 flags;
    UInt8 warningLevel;
    UInt8 reserved;
    UInt8 batteryLevel;
};
typedef struct BatteryInfo BatteryInfo;
```

Fields

`flags`
An unsigned, 8-bit integer representing battery state information.

`warningLevel`
An unsigned, 8-bit integer representing a scaled warning level. The value of this field is in the range of 0-255.

`reserved`
This field is reserved for internal use.

`batteryLevel`
An unsigned, 8-bit integer representing a scaled battery level. The value for this field is in the range of 0-255.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Power.h

BatteryTimeRec

```
struct BatteryTimeRec {
    unsigned long expectedBatteryTime;
    unsigned long minimumBatteryTime;
    unsigned long maximumBatteryTime;
    unsigned long timeUntilCharged;
};
typedef struct BatteryTimeRec BatteryTimeRec;
```

Fields

`expectedBatteryTime`
An unsigned long representing in seconds, the estimated battery time remaining.

`minimumBatteryTime`
An unsigned long representing in seconds, the minimum battery time remaining.

`maximumBatteryTime`
An unsigned long representing in seconds, the maximum battery time remaining.

`timeUntilCharged`
An unsigned long representing in seconds, the time remaining until the battery is fully charged.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Power.h`

DevicePowerInfo

```
struct DevicePowerInfo {
    UInt32 version;
    RegEntryID regID;
    OptionBits flags;
    UInt32 minimumWakeTime;
    UInt32 sleepPowerNeeded;
};
typedef struct DevicePowerInfo DevicePowerInfo;
```

Fields

`version`
The version of this structure.

`regID`
The Registry Entry ID for the device.

`flags`
A value of type `OptionBits` representing device power information.

`minimumWakeTime`
The minimum number of seconds before the device sleeps again.

`sleepPowerNeeded`
The milliwatts the device requires in the sleep state.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Power.h`

HDQueueElement

```
struct HDQueueElement {
    struct HDQueueElement * hdQLink;
    short hdQType;
    short hdFlags;
    HDSpindownUPP hdProc;
    long hdUser;
};
typedef struct HDQueueElement HDQueueElement;
```

Fields

`hdQLink`
A pointer to the next queue element.

`hdQType`
A value of type short representing the queue element type.

`hdFlags`
A value of type short representing flags.

`hdProc`
A pointer to the hard drive spindown routine to call.

`hdUser`
A user-defined value.

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

Declared In
`Power.h`

HDSpindownUPP

```
typedef HDSpindownProcPtr HDSpindownUPP;
```

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

Declared In
`Power.h`

ModemByte

```
typedef SInt8 ModemByte;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Power.h`

PMgrQueueElement

```
struct PMgrQueueElement {
    struct PMgrQueueElement * pmQLink;
    short pmQType;
    short pmFlags;
    long pmNotifyBits;
    PMgrStateChangeUPP pmProc;
    long pmUser;
};
typedef struct PMgrQueueElement PMgrQueueElement;
```

Fields
`pmQLink`
A pointer to the next queue element.

`pmQType`
A value of type short representing the queue element type.

`pmFlags`
A value of type short representing flags.

`pmNotifyBits`
A bitmap representing the changes of which you wish to be notified.

`pmProc`
A pointer to the routine to call.

`pmUser`
A user-defined value.

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

Declared In
`Power.h`

PMgrStateChangeUPP

```
typedef PMgrStateChangeProcPtr PMgrStateChangeUPP;
```

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

Declared In
`Power.h`

PMResultCode

```
typedef long PMResultCode;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Power.h`

PowerLevel

```
typedef UInt32 PowerLevel;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Power.h`

PowerSourceID

```
typedef SInt16 PowerSourceID;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`Power.h`

PowerSourceParamBlock

```
struct PowerSourceParamBlock {
    PowerSourceID sourceID;
    UInt16 sourceCapacityUsage;
    UInt32 sourceVersion;
    OptionBits sourceAttr;
    OptionBits sourceState;
    UInt32 currentCapacity;
    UInt32 maxCapacity;
    UInt32 timeRemaining;
    UInt32 timeToFullCharge;
    UInt32 voltage;
    SInt32 current;
    UInt32 lowWarnLevel;
    UInt32 deadWarnLevel;
    UInt32 reserved[16];
};
typedef struct PowerSourceParamBlock PowerSourceParamBlock;
typedef PowerSourceParamBlock * PowerSourceParamBlockPtr;
```

Fields

`sourceID`
A unique ID assigned by the Power Manager.

`sourceCapacityUsage`
An unsigned, 16-bit integer representing current capacity usage.

`sourceVersion`
An unsigned, 32-bit integer indicating the version of this record.

`sourceAttr`
A value of type `OptionBits` representing power source attributes.

`sourceState`
A value of type `OptionBits` representing power source states.

`currentCapacity`
The current capacity represented in milliwatts or percentage.

`maxCapacity`
The full capacity represented in milliwatts.

`timeRemaining`
The time remaining represented in milliwatt-hours.

`timeToFullCharge`
The time required to charge represented in milliwatt-hours.

`voltage`
The voltage represented in millivolts.

`current`
The current represented in milliamperes. This value may be negative if the power source is consuming.

`lowWarnLevel`
The low warning level represented in milliwatts, or percentage depending on the representation of `sourceCapacityUsage`.

`deadWarnLevel`
The dead warming level represented in milliwatts, or percentage depending on the representation of `sourceCapacityUsage`.

`reserved`
This field is reserved for future expansion.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`Power.h`

PowerSourceParamBlockPtr

```
typedef PowerSourceParamBlock* PowerSourceParamBlockPtr;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`Power.h`

PowerSummary

```
struct PowerSummary {
    UInt32 version;
    OptionBits flags;
    UInt32 sleepPowerAvailable;
    UInt32 sleepPowerNeeded;
    UInt32 minimumWakeTime;
    ItemCount deviceCount;
    DevicePowerInfo devices[1];
};
typedef struct PowerSummary PowerSummary;
```

Fields
`version`
An unsigned, 32-bit integer indicating the version of this record.

`flags`
A value of type `OptionBits` representing power summary information.

`sleepPowerAvailable`
An unsigned, 32-bit integer indicating the milliwatts available during sleep.

`sleepPowerNeeded`
: An unsigned, 32-bit integer indicating the milliwatts needed during sleep.

`minimumWakeTime`
: An unsigned, 32-bit integer indicating the minimum number of seconds required before sleeping again.

`deviceCount`
: The number of device power info records.

`devices`
: An array of device power info records.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Power.h`

SleepQRec

```
struct SleepQRec {
    SleepQRecPtr sleepQLink;
    short sleepQType;
    SleepQUPP sleepQProc;
    short sleepQFlags;
};
typedef struct SleepQRec SleepQRec;
typedef SleepQRec * SleepQRecPtr;
```

Fields
`sleepQLink`
: A pointer to the next element in the queue. This pointer is maintained by the Power Manager; your application should not modify this field.

`sleepQType`
: A short indicating the type of the queue, which must be the constant `sleepQType` (16).

`sleepQProc`
: A pointer to the routine that you provide.

`sleepQFlags`
: A short containing flags which is reserved for use by Apple Computer, Inc.

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

Declared In
`Power.h`

SleepQRecPtr

```
typedef SleepQRec *SleepQRecPtr;
```

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

Declared In
`Power.h`

SleepQUPP

```
typedef SleepQProcPtr SleepQUPP;
```

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

Declared In
`Power.h`

SoundMixerByte

```
typedef SInt8 SoundMixerByte;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Power.h`

StartupTime

```
struct StartupTime {
    unsigned long startTime;
    Boolean startEnabled;
    SInt8 filler;
};
typedef struct StartupTime StartupTime;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Power.h`

WakeupTime

```
struct WakeupTime {
    unsigned long wakeTime;
    Boolean wakeEnabled;
    SInt8 filler;
};
typedef struct WakeupTime WakeupTime;
```

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

Not available to 64-bit applications.

Declared In
`Power.h`

Constants

Apple Event Types and Errors

```
enum {
    kAEMacPowerMgtEvt = 'pmgt',
    kAEMacToWake = 'wake',
    kAEMacLowPowerSaveData = 'pmsd',
    kAEMacEmergencySleep = 'emsl',
    kAEMacEmergencyShutdown = 'emsd'
};
```

BatteryByte Bits

```
enum {
    chargerConnBit = 0,
    hiChargeBit = 1,
    chargeOverFlowBit = 2,
    batteryDeadBit = 3,
    batteryLowBit = 4,
    connChangedBit = 5
};
```

Constants

`chargerConnBit`
When this bit is set, it indicates the charger is connected.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`hiChargeBit`
When this bit is set, it indicates charging at fastest rate.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`chargeOverFlowBit`
When this bit is set, it indicates the hicharge counter has overflowed.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`batteryDeadBit`
Always 0.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`batteryLowBit`
When this bit is set, it indicates the battery is low.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`connChangedBit`
When this bit is set, it indicates the charger connection has changed.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

BatteryByte Masks

```
enum {
    chargerConnMask = 0x01,
    hiChargeMask = 0x02,
    chargeOverFlowMask = 0x04,
    batteryDeadMask = 0x08,
    batteryLowMask = 0x10,
    connChangedMask = 0x20
};
```

Constants

`chargerConnMask`
The charger is connected.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`hiChargeMask`
Charging at fastest rate.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`chargeOverFlowMask`
The hicharge counter has overflowed.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`batteryDeadMask`
The battery is dead.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`batteryLowMask`
The battery is low.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`connChangedMask`
The connection has changed.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

BatteryInfo Bits

```
enum {
    batteryInstalled = 7,
    batteryCharging = 6,
    chargerConnected = 5,
    upsConnected = 4,
    upsIsPowerSource = 3
};
```

Constants

`batteryInstalled`
When this bit is set, it indicates the battery is currently connected.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`batteryCharging`
When this bit is set, it indicates the battery is being charged.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`chargerConnected`
When this bit is set, it indicates the charger is connected.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`upsConnected`

When this bit is set, it indicates there is an uninterruptable power source (UPS) connected.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`upsIsPowerSource`

When this bit is set, it indicates the UPS is the source of power.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

Client Notification Bits

```
enum {
    pmSleepTimeoutChanged = 0,
    pmSleepEnableChanged = 1,
    pmHardDiskTimeoutChanged = 2,
    pmHardDiskSpindownChanged = 3,
    pmDimmingTimeoutChanged = 4,
    pmDimmingEnableChanged = 5,
    pmDiskModeAddressChanged = 6,
    pmProcessorCyclingChanged = 7,
    pmProcessorSpeedChanged = 8,
    pmWakeupTimerChanged = 9,
    pmStartupTimerChanged = 10,
    pmHardDiskPowerRemovedbyUser = 11,
    pmChargeStatusChanged = 12,
    pmPowerLevelChanged = 13,
    pmWakeOnNetActivityChanged = 14
};
```

Client Notification Masks

```
enum {
    pmSleepTimeoutChangedMask = (1 << pmSleepTimeoutChanged),
    pmSleepEnableChangedMask = (1 << pmSleepEnableChanged),
    pmHardDiskTimeoutChangedMask = (1 << pmHardDiskTimeoutChanged),
    pmHardDiskSpindownChangedMask = (1 << pmHardDiskSpindownChanged),
    pmDimmingTimeoutChangedMask = (1 << pmDimmingTimeoutChanged),
    pmDimmingEnableChangedMask = (1 << pmDimmingEnableChanged),
    pmDiskModeAddressChangedMask = (1 << pmDiskModeAddressChanged),
    pmProcessorCyclingChangedMask = (1 << pmProcessorCyclingChanged),
    pmProcessorSpeedChangedMask = (1 << pmProcessorSpeedChanged),
    pmWakeupTimerChangedMask = (1 << pmWakeupTimerChanged),
    pmStartupTimerChangedMask = (1 << pmStartupTimerChanged),
    pmHardDiskPowerRemovedbyUserMask = (1 << pmHardDiskPowerRemovedbyUser),
    pmChargeStatusChangedMask = (1 << pmChargeStatusChanged),
    pmPowerLevelChangedMask = (1 << pmPowerLevelChanged),
    pmWakeOnNetActivityChangedMask = (1 << pmWakeOnNetActivityChanged)
};
```

DevicePowerInfo Flags

```
enum {
    kDevicePCIPowerOffAllowed = (1L << 0),
    kDeviceSupportsPMIS = (1L << 1),
    kDeviceCanAssertPMEDuringSleep = (1L << 2),
    kDeviceUsesCommonLogicPower = (1L << 3),
    kDeviceDriverPresent = (1L << 4),
    kDeviceDriverSupportsPowerMgt = (1L << 5)
};
```

Constants

`kDevicePCIPowerOffAllowed`
If the bit specified by this mask is set, PCI power off is allowed for this device.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`kDeviceSupportsPMIS`
If the bit specified by this mask is set, the device supports Power Manager Interface Specifications.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`kDeviceCanAssertPMEDuringSleep`
If the bit specified by this mask is set, the device can assert the PME# line during sleep.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`kDeviceUsesCommonLogicPower`
If the bit specified by this mask is set, the device uses common-logic power.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`kDeviceDriverPresent`
If the bit specified by this mask is set, the device driver is present.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`kDeviceDriverSupportsPowerMgt`
If the bit specified by this mask is set, the device driver installed a power handler.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

HDPwrQType Constants

```
enum {
    HDPwrQType = 0x4844,
    PMgrStateQType = 0x504D
};
```

Constants

`HDPwrQType`
The hard disk spindown queue element type.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`PMgrStateQType`
The Power Manager state queue element type.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

HDQueueElement Flags

```
enum {
    kHDQueuePostBit = 0,
    kHDQueuePostMask = (1 << kHDQueuePostBit)
};
```

Constants

`kHDQueuePostBit`
When this bit is set, it indicates the routine will be called on the second pass.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`kHDQueuePostMask`
If the bit specified by this mask is set, it indicates the routine will be called on the second pass.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

kMediaPowerCSCode Constants

```
enum {
    kMediaPowerCSCode = 70
};
```

kUseDefaultMinimumWakeTime Constants

```
enum {
    kUseDefaultMinimumWakeTime = 0,
    kPowerSummaryVersion = 1,
    kDevicePowerInfoVersion = 1
};
```

Constants

`kUseDefaultMinimumWakeTime`
Defaults to 5 minutes.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`kPowerSummaryVersion`
Version of `PowerSummary` structure.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`kDevicePowerInfoVersion`
Version of `DevicePowerInfo` structure

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

Modem State Bits

```
enum {
    hasInternalModem = 0,
    intModemRingDetect = 1,
    intModemOffHook = 2,
    intModemRingWakeEnb = 3,
    extModemSelected = 4,
    modemSetBit = 15
};
```

Constants

`hasInternalModem`
When this bit is set, it indicates an internal modem is installed.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`intModemRingDetect`
When this bit is set, it indicates the internal modem has detected a ring.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`intModemOffHook`
When this bit is set, it indicates the internal modem is off the hook.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`intModemRingWakeEnb`
When this bit is set, it indicates wakeup on ring is enabled.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`extModemSelected`
When this bit is set, it indicates external modem is selected.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`modemSetBit`
When this bit is set, it indicates set bit. If 0, clear bit.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

Discussion
These bits are in the bit field returned by the `GetIntModemInfo` (page 1600) function and set by the `SetIntModemState` (page 1615) function.

ModemByte Bits

```
enum {
    modemOnBit = 0,
    ringWakeUpBit = 2,
    modemInstalledBit = 3,
    ringDetectBit = 4,
    modemOnHookBit = 5
};
```

Constants
`modemOnBit`
When this bit is set, it indicates the modem is on.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`ringWakeUpBit`
When this bit is set, it indicates ring wakeup is enabled.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`modemInstalledBit`
When this bit is set, it indicates an internal modem is installed.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`ringDetectBit`
When this bit is set, it indicates an incoming call is detected.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`modemOnHookBit`
When this bit is set, it indicates the modem is off the hook.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

ModemByte Masks

```
enum {
    modemOnMask = 0x01,
    ringWakeUpMask = 0x04,
    modemInstalledMask = 0x08,
    ringDetectMask = 0x10,
    modemOnHookMask = 0x20
};
```

Constants

`modemOnMask`
The modem is on.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`ringWakeUpMask`
Ring wakeup is enabled.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`modemInstalledMask`
An internal modem is installed.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`ringDetectMask`
An incoming call is detected.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`modemOnHookMask`
The modem is off the hook.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

Net Activity Wake Options

```
enum {
    kConfigSupportsWakeOnNetBit = 0,
    kWakeOnNetAdminAccessesBit = 1,
    kWakeOnAllNetAccessesBit = 2,
    kUnmountServersBeforeSleepingBit = 3,
    kConfigSupportsWakeOnNetMask = (1 << kConfigSupportsWakeOnNetBit),
    kWakeOnNetAdminAccessesMask = (1 << kWakeOnNetAdminAccessesBit),
    kWakeOnAllNetAccessesMask = (1 << kWakeOnAllNetAccessesBit),
    kUnmountServersBeforeSleepingMask = (1 << kUnmountServersBeforeSleepingBit)
};
```

PCI Bus PMIS Power Levels

```
enum {
    kPMDevicePowerLevel_On = 0,
    kPMDevicePowerLevel_D1 = 1,
    kPMDevicePowerLevel_D2 = 2,
    kPMDevicePowerLevel_Off = 3
};
```

Constants

`kPMDevicePowerLevel_On`

When this bit is set, it indicates the PCI bus is fully powered.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`kPMDevicePowerLevel_D1`

Reserved.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`kPMDevicePowerLevel_D2`

Reserved.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`kPMDevicePowerLevel_Off`

When this bit is set, it indicates the main PCI bus power is off, but PCI standby power is available.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

Power Capacity Types

```
enum {
    kCapacityIsActual = 0,
    kCapacityIsPercentOfMax = 1
};
```

Constants

`kCapacityIsActual`

The capacity is expressed as actual capacity in the same units as `maxCapacity`.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

`kCapacityIsPercentOfMax`

The capacity is expressed as a percentage of `maxCapacity`.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

Power Handler Wake Results

```
enum {
    kDeviceDidNotWakeMachine = 0,
    kDeviceRequestsFullWake = 1,
    kDeviceRequestsWakeToDoze = 2
};
```

Constants

`kDeviceDidNotWakeMachine`

The device did not wake the computer.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`kDeviceRequestsFullWake`

The device did wake the computer and requests full wakeup.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`kDeviceRequestsWakeToDoze`

The device did wake the computer and requests partial wakeup.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

Discussion

On query by the Power Manager, these are result values returned by a power handler if the device the power handler represents woke the computer.

Power Manager Features Bits

```
enum {
    hasWakeupTimer = 0,
    hasSharedModemPort = 1,
    hasProcessorCycling = 2,
    mustProcessorCycle = 3,
    hasReducedSpeed = 4,
    dynamicSpeedChange = 5,
    hasSCSIDiskMode = 6,
    canGetBatteryTime = 7,
    canWakeupOnRing = 8,
    hasDimmingSupport = 9,
    hasStartupTimer = 10,
    hasChargeNotification = 11,
    hasDimSuspendSupport = 12,
    hasWakeOnNetActivity = 13,
    hasWakeOnLid = 14,
    canPowerOffPCIBus = 15,
    hasDeepSleep = 16,
    hasSleep = 17,
    supportsServerModeAPIs = 18,
    supportsUPSIntegration = 19,
    hasAggressiveIdling = 20,
    supportsIdleQueue = 21
};
```

Constants

`hasWakeupTimer`

When this bit is set, it indicates the wakeup timer is supported.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`hasSharedModemPort`

When this bit is set, it indicates the modem port is shared by the serial communications chip (SCC) and the internal modem.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`hasProcessorCycling`

When this bit is set, it indicates processor cycling is supported.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`mustProcessorCycle`

When this bit is set, it indicates processor cycling should not be turned off.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`hasReducedSpeed`

When this bit is set, it indicates the processor can be started up at a reduced speed.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`dynamicSpeedChange`

When this bit is set, it indicates the processor speed can be switched dynamically.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`hasSCSIDiskMode`

When this bit is set, it indicates SCSI disk mode is supported.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`canGetBatteryTime`

When this bit is set, it indicates battery time can be calculated.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`canWakeupOnRing`

When this bit is set, it indicates wakeup when the modem detects a ring.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`hasDimmingSupport`

When this bit is set, it indicates dimming support is built in—display power management system (DPMS) standby by default.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`hasStartupTimer`

When this bit is set, it indicates the startup timer is supported.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`hasChargeNotification`

When this bit is set, it indicates the client can determine charge connect status change notification available.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`hasDimSuspendSupport`
When this bit is set, it indicates support of dimming LCD and CRT to DPMS suspend state.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`hasWakeOnNetActivity`
When this bit is set, it indicates hardware supports wake on network activity.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`hasWakeOnLid`
When this bit is set, it indicates hardware can wake when opened.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`canPowerOffPCIBus`
When this bit is set, it indicates hardware can power off PCI bus during sleep if cards allow.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`hasDeepSleep`
When this bit is set, it indicates hardware supports deep sleep (hibernation) mode.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`hasSleep`
When this bit is set, it indicates hardware supports normal sleep.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`supportsServerModeAPIs`
When this bit is set, it indicates hardware supports server mode API.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`supportsUPSIntegration`
When this bit is set, it indicates hardware supports UPS integration and reporting.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`hasAggressiveIdling`
When this bit is set, it indicates Power Manager only resets `OverallAct` on `UsrActivity`.

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`supportsIdleQueue`
When this bit is set, it indicates Power Manager supports the idle queue.

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

Discussion
These bits are in the bit field returned by the `PMFeatures` (page 1612) function.

Power Source Attribute Bits

```
enum {
    bSourceIsBattery = 0,
    bSourceIsAC = 1,
    bSourceCanBeCharged = 2,
    bSourceIsUPS = 3,
    bSourceProvidesWarnLevels = 4,
    kSourceIsBatteryMask = (1 << bSourceIsBattery),
    kSourceIsACMask = (1 << bSourceIsAC),
    kSourceCanBeChargedMask = (1 << bSourceCanBeCharged),
    kSourceIsUPSMask = (1 << bSourceIsUPS),
    kSourceProvidesWarnLevelsMask = (1 << bSourceProvidesWarnLevels)
};
```

Constants
`bSourceIsBattery`
When this bit is set, it indicates the power source is a battery.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

`bSourceIsAC`
When this bit is set, it indicates the power source is AC.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

`bSourceCanBeCharged`
When this bit is set, it indicates the power source can be charged.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

`bSourceIsUPS`
When this bit is set, it indicates the power source is an uninterruptable power supply (UPS). `bSourceIsBattery` and `bSourceIsAC` should be set as well if appropriate.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

`bSourceProvidesWarnLevels`
When this bit is set, it indicates power source provides low power and dead battery warning levels.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

`kSourceIsBatteryMask`
If the bit specified by this mask is set, the power source is a battery.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

`kSourceIsACMask`
If the bit specified by this mask is set, the power source is AC.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

`kSourceCanBeChargedMask`
If the bit specified by this mask is set, the power source can be charged.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

`kSourceIsUPSMask`
If the bit specified by this mask is set, the power source is a UPS.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

`kSourceProvidesWarnLevelsMask`
If the bit specified by this mask is set, the power source provides low power and dead battery warning levels.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

Power Source Capacity Usage Types

```
enum {
    kCurrentCapacityIsActualValue = 0,
    kCurrentCapacityIsPercentOfMax = 1
};
```

Constants

`kCurrentCapacityIsActualValue`
The current capacity is expressed as a real value in the same units as `maxCapacity`.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

`kCurrentCapacityIsPercentOfMax`
The current capacity is expressed as a percentage of `maxCapacity`.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

Power Source State Bits

```
enum {
    bSourceIsAvailable = 0,
    bSourceIsCharging = 1,
    bChargerIsAttached = 2,
    kSourceIsAvailableMask = (1 << bSourceIsAvailable),
    kSourceIsChargingMask = (1 << bSourceIsCharging),
    kChargerIsAttachedMask = (1 << bChargerIsAttached)
};
```

Constants

`bSourceIsAvailable`
When this bit is set, it indicates a power source is installed.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

`bSourceIsCharging`
When this bit is set, it indicates a power source is charging.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

`bChargerIsAttached`
When this bit is set, it indicates a charger is connected.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

`kSourceIsAvailableMask`
If the bit specified by this mask is set, the power source is installed.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

`kSourceIsChargingMask`
If the bit specified by this mask is set, the power source is charging.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

`kChargerIsAttachedMask`
If the bit specified by this mask is set, a charger is connected.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `Power.h`.

Power Source Version

```
enum {
    kVersionOnePowerSource = 1,
    kVersionTwoPowerSource = 2,
    kCurrentPowerSourceVersion = kVersionTwoPowerSource
};
```

Power Summary Flags

```
enum {
    kPCIPowerOffAllowed = (1L << 0)
};
```

Constants

`kPCIPowerOffAllowed`

If the bit specified by this mask is set, it indicates PCI power off is allowed.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

Sleep Commands

Not recommended

```
enum {
    sleepRequest = kSleepRequest,
    sleepDemand = kSleepDemand,
    sleepWakeUp = kSleepWakeUp,
    sleepRevoke = kSleepRevoke,
    sleepUnlock = kSleepUnlock,
    sleepDeny = kSleepDeny,
    sleepNow = kSleepNow,
    dozeDemand = kDozeDemand,
    dozeWakeUp = kDozeWakeUp,
    dozeRequest = kDozeRequest,
    enterStandby = kEnterStandby,
    enterRun = kEnterRun,
    suspendRequestMsg = kSuspendRequest,
    suspendDemandMsg = kSuspendDemand,
    suspendRevokeMsg = kSuspendRevoke,
    suspendWakeUpMsg = kSuspendWakeUp,
    getPowerLevel = kGetPowerLevel,
    setPowerLevel = kSetPowerLevel
};
```

Constants

`sleepRequest`

A sleep request.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`sleepDemand`
A sleep demand.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`sleepWakeUp`
A wakeup demand.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`sleepRevoke`
A sleep request revocation.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

sleepQFlags Bits

```
enum {
    noCalls = 1,
    noRequest = 2,
    slpQType = 16,
    sleepQType = 16
};
```

Constants

`noCalls`
A `noCalls` queue type.

Available in Mac OS X v10.0 and later.

Declared in `Power.h`.

`noRequest`
A `noRequest` queue type.

Available in Mac OS X v10.0 and later.

Declared in `Power.h`.

`slpQType`
A `sleepQType` queue.

Available in Mac OS X v10.0 and later.

Declared in `Power.h`.

`sleepQType`
A `sleepQType` queue.

Available in Mac OS X v10.0 and later.

Declared in `Power.h`.

sleepQProc Commands

```
enum {
    kSleepRequest = 1,
    kSleepDemand = 2,
    kSleepWakeUp = 3,
    kSleepRevoke = 4,
    kSleepUnlock = 4,
    kSleepDeny = 5,
    kSleepNow = 6,
    kDozeDemand = 7,
    kDozeWakeUp = 8,
    kDozeRequest = 9,
    kEnterStandby = 10,
    kEnterRun = 11,
    kSuspendRequest = 12,
    kSuspendDemand = 13,
    kSuspendRevoke = 14,
    kSuspendWakeUp = 15,
    kGetPowerLevel = 16,
    kSetPowerLevel = 17,
    kDeviceInitiatedWake = 18,
    kWakeToDoze = 19,
    kDozeToFullWakeUp = 20,
    kGetPowerInfo = 21,
    kGetWakeOnNetInfo = 22,
    kSuspendWakeToDoze = 23,
    kEnterIdle = 24,
    kStillIdle = 25,
    kExitIdle = 26
};
```

Constants

`kSleepDeny`

A non-zero value clients can use to deny requests.

Available in Mac OS X v10.0 and later.

Declared in `Power.h`.

`kDozeRequest`

Additional messages for Power Manager 2.0.

Available in Mac OS X v10.0 and later.

Declared in `Power.h`.

`kEnterStandby`

Idle queue only.

Available in Mac OS X v10.0 and later.

Declared in `Power.h`.

`kEnterRun`

Idle queue only.

Available in Mac OS X v10.0 and later.

Declared in `Power.h`.

`kEnterIdle`
Idle queue only.
Available in Mac OS X v10.1 and later.
Declared in `Power.h`.

`kStillIdle`
Idle queue only.
Available in Mac OS X v10.1 and later.
Declared in `Power.h`.

`kExitIdle`
Idle queue only.
Available in Mac OS X v10.1 and later.
Declared in `Power.h`.

SoundMixerByte Bits

```
enum {
    MediaBaySndEnBit = 0,
    PCISndEnBit = 1,
    ZVSndEnBit = 2,
    PCCardSndEnBit = 3
};
```

SoundMixerByte Masks

```
enum {
    MediaBaySndEnMask = 0x01,
    PCISndEnMask = 0x02,
    ZVSndEnMask = 0x04,
    PCCardSndEnMask = 0x08
};
```

Storage Media Sleep Modes

```
enum {
    kMediaModeOn = 0,
    kMediaModeStandBy = 1,
    kMediaModeSuspend = 2,
    kMediaModeOff = 3
};
```

Constants

`kMediaModeOn`
When this bit is set, it indicates the media is active—the drive is spinning at full power.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Power.h`.

`kMediaModeStandBy`
When this bit is set, it indicates the media is on standby. This is not implemented.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`kMediaModeSuspend`
When this bit is set, it indicates the media is idle. This is not implemented.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

`kMediaModeOff`
When this bit is set, it indicates the media is asleep—the drive is not spinning and is at minimum power and maximum recovery time.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Power.h`.

System Activity Selectors

```
enum {
    OverallAct = 0,
    UsrActivity = 1,
    NetActivity = 2,
    HDActivity = 3,
    IdleActivity = 4
};
```

Constants

`OverallAct`
Delays idle sleep by a small amount. This will only delay power cycling if it's enabled, and will delay sleep by a small amount when `hasAggressiveIdling` is set.

Available in Mac OS X v10.0 and later.

Declared in `Power.h`.

`UsrActivity`
Delays idle sleep and dimming by timeout time.

Available in Mac OS X v10.0 and later.

Declared in `Power.h`.

`NetActivity`
Delays idle sleep and power cycling by small amount.

Available in Mac OS X v10.0 and later.

Declared in `Power.h`.

`HDActivity`
Delays hard drive spindown and idle sleep by small amount.

Available in Mac OS X v10.0 and later.

Declared in `Power.h`.

`IdleActivity`

Delays idle sleep by timeout time. The `IdleActivity` selector is not available unless the `hasAggressiveIdling` bit is set. Use `IdleActivity` where you used to use `OverallAct` if necessary. Don't use `IdleActivity` unless `hasAggressiveIdling` is set; when `hasAggressiveIdling` is not set, the use of `IdleActivity` is undefined, and will do different things depending on which Power Manager is currently running.

Available in Mac OS X v10.1 and later.

Declared in `Power.h`.

Result Codes

The most common result codes returned by Power Manager are listed below.

Result Code	Value	Description
`noErr`	0	No error Available in Mac OS X v10.0 and later.
`pmBusyErr`	-13000	Power Manager IC stuck busy Available in Mac OS X v10.0 and later.
`pmReplyTOErr`	-13001	Timed out waiting to begin reply handshake Available in Mac OS X v10.0 and later.
`pmSendStartErr`	-13002	Power Manager IC did not start handshake Available in Mac OS X v10.0 and later.
`pmSendEndErr`	-13003	During send, Power Manager did not finish handshake Available in Mac OS X v10.0 and later.
`pmRecvStartErr`	-13004	During receive, Power Manager did not start handshake Available in Mac OS X v10.0 and later.
`pmRecvEndErr`	-13005	During receive, Power Manager did not finish handshake Available in Mac OS X v10.0 and later.

Resource Manager Reference

Framework: CoreServices/CoreServices.h

Declared in Resources.h

Overview

The Resource Manager allows applications to create, delete, open, read, modify, and write resources, get information about them, and alter the Resource Manager's search path. A resource is data of any kind stored in a defined format in a resource file. The Resource Manager keeps track of resources in memory and provides functions for the proper management of the resource chain. In Mac OS X, you should store resources in the data fork of a resource file.

Carbon applications have used Resource Manager resources to store the descriptions for user interface elements such as menus, windows, dialogs, controls, and icons. In addition, applications have used resources to store variable settings, such as the location of a document window at the time the user closes the window. When the user opens the document again, the application reads the information in the appropriate resource and restores the window to its previous location.

Functions by Task

Checking for Errors

`ResError` (page 1692)
Determines what error occurred, if any, after calling a Resource Manager function.

Closing Resource Forks

`CloseResFile` (page 1662)
Closes a resource fork before your application terminates.

Counting and Listing Resource Types

`Count1Resources` (page 1663)
Returns the total number of resources of a given type in the current resource file.

`Count1Types` (page 1663)
Returns the number of resource types in the current resource file.

`CountResources` (page 1664)
Returns the total number of available resources of a given type.

`CountTypes` (page 1664)
Returns the number of resource types in all resource forks open to your application.

`Get1IndResource` (page 1675)
Returns a handle to a resource of a given type in the current resource file.

`Get1IndType` (page 1675)
Gets a resource types available in the current resource file.

`GetIndResource` (page 1678)
Returns a handle to a resource of a given type in resource forks open to your application.

`GetIndType` (page 1678)
Gets a resource type available in resource forks open to your application.

Creating Resource Files and Forks

`FSCreateResourceFile` (page 1667)
Creates a file with a named fork for storing resource data.

`FSCreateResourceFork` (page 1668)
Creates a named fork for storing resource data.

`FSCreateResFile` (page 1666)
Creates a file with an empty resource fork.

`FSpCreateResFile` (page 1670) Deprecated in Mac OS X v10.5
Creates an empty resource fork in a new or existing file. (Deprecated. Use `FSCreateResourceFile` (page 1667) instead.)

`HCreateResFile` (page 1685) Deprecated in Mac OS X v10.5
Creates an empty resource fork, when the `FSpCreateResFile` function is not available. (Deprecated. Use `FSCreateResourceFile` (page 1667) instead.)

Disposing of Resources

`DetachResource` (page 1665)
Sets the value of a resource's handle in the resource map in memory to `NULL` while keeping the resource data in memory.

`ReleaseResource` (page 1691)
Releases the memory a resource occupies when you have finished using it.

`RemoveResource` (page 1692)
Removes a resource's entry from the current resource file's resource map in memory.

Getting a Unique Resource ID

`Unique1ID` (page 1698)
Gets a resource ID that's unique with respect to resources in the current resource file.

UniqueID (page 1699)
Gets a unique resource ID for a resource.

Getting and Setting Resource Fork Attributes

GetResFileAttrs (page 1682)
Gets the attributes of a resource fork.

SetResFileAttrs (page 1694)
Sets a resource fork's attributes.

Getting and Setting Resource Information

GetResAttrs (page 1681)
Gets a resource's attributes.

GetResInfo (page 1682)
Gets a resource's resource ID, resource type, and resource name.

SetResAttrs (page 1693)
Sets a resource's attributes in the resource map in memory.

SetResInfo (page 1694)
Sets the name and resource ID of a resource.

Getting and Setting the Current Resource File

CurResFile (page 1664)
Gets the file reference number of the current resource file.

HomeResFile (page 1685)
Gets the file reference number associated with a particular resource.

UseResFile (page 1700)
Sets the current resource file.

Getting Resource Sizes

GetMaxResourceSize (page 1679)
Returns the approximate size of a resource.

GetResourceSizeOnDisk (page 1684)
Returns the exact size of a resource.

Managing the Resource Chain

InsertResourceFile (page 1687)
Inserts a resource file into the current resource chain at the specified location.

`DetachResourceFile` (page 1666)
Removes a resource file from the resource chain.

`GetNextResourceFile` (page 1681)
Retrieves the next resource file in the resource chain.

`GetTopResourceFile` (page 1684)
Retrieves the topmost resource file in the current resource chain.

Modifying Resources

`AddResource` (page 1660)
Adds a resource to the current resource file's resource map in memory.

`ChangedResource` (page 1661)
Sets a flag in the resource's resource map entry in memory to show that you've made changes to a resource's data or to an entry in a resource map.

Opening Resource Forks

`FSOpenResourceFile` (page 1670)
Opens a named fork in an existing resource file.

`FSOpenOrphanResFile` (page 1669)
Opens a resource file that is persistent across all contexts.

`FSOpenResFile` (page 1669)
Opens the resource fork in a file specified with an `FSRef` structure.

`FSResourceFileAlreadyOpen` (page 1674)
Checks whether a resource file is open.

`FSpOpenOrphanResFile` (page 1671) Deprecated in Mac OS X v10.5
Opens a resource file that is persistent across all contexts. (Deprecated. Use `FSOpenOrphanResFile` (page 1669) instead.)

`FSpOpenResFile` (page 1671) Deprecated in Mac OS X v10.5
Opens the resource fork in a file specified with an `FSSpec` structure. (Deprecated. Use `FSOpenResourceFile` (page 1670) instead.)

`FSpResourceFileAlreadyOpen` (page 1673) Deprecated in Mac OS X v10.5
Checks whether a resource file is open. (Deprecated. Use `FSResourceFileAlreadyOpen` (page 1674) instead.)

`HOpenResFile` (page 1686) Deprecated in Mac OS X v10.5
Opens a file's resource fork, when the `FSpOpenResFile` function is not available. (Deprecated. Use `FSOpenResourceFile` (page 1670) instead.)

`OpenRFPerm` (page 1689) Deprecated in Mac OS X v10.5
Opens a file's resource fork, when the `FSpOpenResFile` and `HOpenResFile` functions are not available. (Deprecated. Use `FSOpenResourceFile` (page 1670) instead.)

Reading and Writing Partial Resources

`ReadPartialResource` (page 1690)
Reads part of a resource into memory and work with a small subsection of a large resource.

`SetResourceSize` (page 1696)
Sets the size of a resource on disk.

`WritePartialResource` (page 1701)
Writes part of a resource to disk when working with a small subsection of a large resource.

Reading Resources Into Memory

`Get1NamedResource` (page 1676)
Gets a named resource in the current resource file.

`Get1Resource` (page 1677)
Gets resource data for a resource in the current resource file.

`GetNamedResource` (page 1680)
Gets a named resource.

`GetResource` (page 1683)
Gets resource data for a resource specified by resource type and resource ID.

`LoadResource` (page 1688)
Gets resource data after you've called the `SetResLoad` function with the `load` parameter set to `FALSE` or when the resource is purgeable.

`SetResLoad` (page 1695)
Enables and disables automatic loading of resource data into memory for functions that return handles to resources.

Writing to Resource Forks

`SetResPurge` (page 1697)
Tells the Memory Manager to pass the handle of a resource to the Resource Manager before purging the data specified by that handle.

`UpdateResFile` (page 1700)
Updates the resource map and resource data for a resource fork without closing it.

`WriteResource` (page 1702)
Writes resource data in memory immediately to a file's resource fork.

Not Recommended

`GetNextFOND` (page 1680)
Gets the next FOND handle.

`InvokeResErrUPP` (page 1688)
Calls your callback function.

`NewResErrUPP` (page 1688)
Creates a new universal procedure pointer (UPP) to your callback function.

`DisposeResErrUPP` (page 1666)
Disposes of the universal procedure pointer (UPP) to your callback function.

Functions

AddResource

Adds a resource to the current resource file's resource map in memory.

```
void AddResource (
   Handle theData,
   ResType theType,
   ResID theID,
   ConstStr255Param name
);
```

Parameters

theData
A handle to data in memory to be added as a resource to the current resource file (not a handle to an existing resource). If the value of this parameter is an empty handle (that is, a handle whose master pointer is set to `NULL`), the Resource Manager writes zero-length resource data to disk when it updates the resource. If its value is either `NULL` or a handle to an existing resource, the function does nothing, and the `ResError` (page 1692) function returns the result code `addResFailed`. If the resource map becomes too large to fit in memory, the function does nothing, and `ResError` returns an appropriate result code. The same is true if the resource data in memory can't be written to the resource fork (for example, because the disk is full).

theType
The resource type of the resource to be added.

theID
The resource ID of the resource to be added.

name
The name of the resource to be added.

Discussion

This function sets the `resChanged` attribute to 1 it does not set any of the resource's other attributes—that is, all other attributes are set to 0. If the `resChanged` attribute of a resource has been set and your application calls the `UpdateResFile` (page 1700) function or quits, the Resource Manager writes both the resource map and the resource data for that resource to the resource fork of the corresponding file on disk. If the `resChanged` attribute for a resource has been set and your application calls the `WriteResource` (page 1702) function, the Resource Manager writes only the resource data for that resource to disk.

If you add a resource to the current resource file, the Resource Manager writes the entire resource map to disk when it updates the file. If you want any of your changes to the resource map or resource data to be temporary, you must restore the original information before the Resource Manager updates the file on disk.

The function doesn't verify whether the resource ID you pass in the parameter `theID` is already assigned to another resource of the same type. You should call the `UniqueID` (page 1699) or `Unique1ID` (page 1698) function to get a unique resource ID before adding a resource with this function.

When your application calls this function, the Resource Manager attempts to reserve disk space for the new resource. If the new resource data can't be written to the resource fork (for example, if there's not enough room on disk), the `resChanged` attribute is not set to 1. If this is the case and you call `UpdateResFile` or `WriteResource`, the Resource Manager won't know that resource data has been added. Thus, the function won't write the new resource data to the resource fork and won't return an error. For this reason, always make sure that the `ResError` function returns the result code `noErr` after a call to `AddResource`.

To copy an existing resource, get a handle to the resource you want to copy, call the `DetachResource` (page 1665) function, then call `AddResource`. To add the same resource data to several different resource forks, call the Memory Manager function `HandToHand` to duplicate the handle for each resource.

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

Declared In
`Resources.h`

ChangedResource

Sets a flag in the resource's resource map entry in memory to show that you've made changes to a resource's data or to an entry in a resource map.

```
void ChangedResource (
   Handle theResource
);
```

Parameters

theResource

A handle to the resource whose data you've changed. The function sets the `resChanged` attribute for that resource in the resource map in memory. If the `resChanged` attribute for a resource has been set and your application calls the `UpdateResFile` (page 1700) function or quits, the Resource Manager writes the resource data for that resource (and for all other resources whose `resChanged` attribute is set) and the entire resource map to the resource fork of the corresponding file on disk. If the `resChanged` attribute for a resource has been set and your application calls the `WriteResource` (page 1702) function, the Resource Manager writes only the resource data for that resource to disk.

If the given handle isn't a handle to a resource, if the modified resource data can't be written to the resource fork, or if the `resProtected` attribute is set for the modified resource, the function does nothing. To find out whether any of these errors occurred, call the `ResError` (page 1692) function.

Discussion
If you change information in the resource map with a call to the `SetResInfo` (page 1694) or `SetResAttrs` (page 1693) function and then call `ChangedResource` and `UpdateResFile`, the Resource Manager still writes both the resource map and the resource data to disk. If you want any of your changes to the resource map or resource data to be temporary, you must restore the original information before the Resource Manager updates the resource fork on disk.

After writing a resource to disk, the Resource Manager clears the resource's `resChanged` attribute in the appropriate entry of the resource map in memory.

When your application calls this function, the Resource Manager attempts to reserve enough disk space to contain the changed resource. The function reserves space every time you call it, but the resource is not actually written until you call `WriteResource` or `UpdateResFile`. Thus, if you call `ChangedResource`

several times before the resource is actually written, the function reserves much more space than is needed. If the resource is large, you may unexpectedly run out of disk space. When the resource is actually written, the file's end-of-file (EOF) is set correctly, and the next call to ChangedResource will work as expected.

If the modified resource data can't be written to the resource fork (for example, if there's not enough room on disk), the resChanged attribute is not set to 1. If this is the case and you call UpdateResFile or WriteResource, the Resource Manager won't know that the resource data has been changed. Thus, the function won't write the modified resource data to the resource fork and won't return an error. For this reason, always make sure that the ResError function returns the result code noErr after a call to ChangedResource.

If your application frequently changes the contents of resources (especially large resources), you should call WriteResource or UpdateResFile immediately after calling ChangedResource.

If you need to make changes to a purgeable resource using functions that may cause the Memory Manager to purge the resource, you should make the resource temporarily not purgeable. To do so, use the Memory Manager functions HGetState, HNoPurge, and HSetState to make sure the resource data remains in memory while you change it and until the resource data is written to disk. (You can't use the SetResAttrs function for this purpose, because the changes don't take effect immediately.) First call HGetState and HNoPurge, then change the resource as necessary. To make a change to a resource permanent, use ChangedResource and UpdateResFile or WriteResource; then call HSetState when you're finished. Or, instead of calling WriteResource to write the resource data immediately, you can call the SetResPurge (page 1697) function with the install parameter set to TRUE before making changes to purgeable resource data.

If your application doesn't make its resources purgeable, or if the changes you are making to a purgeable resource don't involve functions that may cause the resource to be purged, you don't need to take these precautions

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

Declared In
Resources.h

CloseResFile

Closes a resource fork before your application terminates.

```
void CloseResFile (
   ResFileRefNum refNum
);
```

Parameters

refNum

The file reference number for the resource fork to close. If this parameter does not contain a file reference number for a file whose resource fork is open, the function does nothing, and the ResError (page 1692) function returns the result code resFNotFound. If the value of this parameter is 0, it represents the System file and is ignored. You cannot close the System file's resource fork.

Discussion
This function performs four tasks. First, it updates the file by calling the UpdateResFile (page 1700) function. Second, it releases the memory occupied by each resource in the resource fork by calling the DisposeHandle function. Third, it releases the memory occupied by the resource map. The fourth task is to close the resource fork.

When your application terminates, the Resource Manager automatically closes every resource fork open to your application except the System file's resource fork. You need to call this function only if you want to close a resource fork before your application terminates.

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

Related Sample Code
Simple DrawSprocket

Declared In
`Resources.h`

Count1Resources

Returns the total number of resources of a given type in the current resource file.

```
ResourceCount Count1Resources (
   ResType theType
);
```

Parameters

theType
The resource type of the resources to count.

Return Value
The total number of resources of the given type in the current resource file.

Discussion
To check for errors, call the `ResError` (page 1692) function.

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

Declared In
`Resources.h`

Count1Types

Returns the number of resource types in the current resource file.

```
ResourceCount Count1Types (
   void
);
```

Return Value
The total number of unique resource types in the current resource file.

Discussion
To check for errors, call the `ResError` (page 1692) function.

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

Declared In
Resources.h

CountResources

Returns the total number of available resources of a given type.

```
ResourceCount CountResources (
   ResType theType
);
```

Parameters

theType
A resource type.

Return Value
The total number of resources of the given type in all resource forks open to your application.

Discussion
To check for errors, call the ResError (page 1692) function.

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

Declared In
Resources.h

CountTypes

Returns the number of resource types in all resource forks open to your application.

```
ResourceCount CountTypes (
   void
);
```

Return Value
The total number of unique resource types in all resource forks open to your application.

Discussion
To check for errors, call the ResError (page 1692) function.

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

Declared In
Resources.h

CurResFile

Gets the file reference number of the current resource file.

```
ResFileRefNum CurResFile (
   void
);
```

Return Value
The file reference number associated with the current resource file. You can call this function when your application starts up (before opening the resource fork of any other file) to get the file reference number of your application's resource fork. If the current resource file is the System file, the function returns the actual file reference number. You can use this number or 0 with functions that take a file reference number for the System file. All Resource Manager functions recognize both 0 and the actual file reference number as referring to the System file.

Discussion
Most of the Resource Manager functions assume that the current resource file is the file on whose resource fork they should operate or, in the case of a search, the file where they should begin. In general, the current resource file is the last one whose resource fork your application opened unless you specify otherwise.

To check for errors, call the `ResError` (page 1692) function.

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

Related Sample Code
Simple DrawSprocket

Declared In
`Resources.h`

DetachResource

Sets the value of a resource's handle in the resource map in memory to `NULL` while keeping the resource data in memory.

```
void DetachResource (
   Handle theResource
);
```

Parameters

`theResource`
A handle to the resource which you wish to detach. If this parameter doesn't contain a handle to a resource or if the resource's `resChanged` attribute is set, the function does nothing. To determine whether either of these errors occurred, call the `ResError` (page 1692) function.

Discussion
After this call, the Resource Manager no longer recognizes the handle as a handle to a resource. However, this function does not release the memory used for the resource data, and the master pointer is still valid. Thus, you can access the resource data directly by using the handle.

If your application subsequently calls a Resource Manager function to get the released resource, the Resource Manager assigns a new handle. You can use `DetachResource` if you want to access the resource data directly without using Resource Manager functions. You can also use the `DetachResource` function to keep resource data in memory after closing a resource fork.

To copy a resource and install an entry for the duplicate in the resource map, call `DetachResource`, then call the `AddResource` (page 1660) function using a different resource ID.

Special Considerations
Do not use this function to detach a System resource that might be shared by several applications.

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

Related Sample Code
QTMetaData

Declared In
`Resources.h`

DetachResourceFile

Removes a resource file from the resource chain.

```
OSErr DetachResourceFile (
   ResFileRefNum refNum
);
```

Return Value
A result code. See "Resource Manager Result Codes" (page 1711).

Discussion
If the file is not currently in the resource chain, this returns `resNotFound`. Otherwise, the resource file is removed from the resource chain.

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

Declared In
`Resources.h`

DisposeResErrUPP

Disposes of the universal procedure pointer (UPP) to your callback function.

```
void DisposeResErrUPP (
    ResErrUPP userUPP
);
```

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

Declared In
`Resources.h`

FSCreateResFile

Creates a file with an empty resource fork.

```
void FSCreateResFile (
    const FSRef *parentRef,
    UniCharCount nameLength,
    const UniChar *name,
    FSCatalogInfoBitmap whichInfo,
    const FSCatalogInfo *catalogInfo,
    FSRef *newRef,
    FSSpec *newSpec
);
```

Discussion
This function is not recommended. You should use a file's data fork instead of its resource fork to store resource data.

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

Declared In
`Resources.h`

FSCreateResourceFile

Creates a file with a named fork for storing resource data.

```
OSErr FSCreateResourceFile (
   const FSRef *parentRef,
   UniCharCount nameLength,
   const UniChar *name,
   FSCatalogInfoBitmap whichInfo,
   const FSCatalogInfo *catalogInfo,
   UniCharCount forkNameLength,
   const UniChar *forkName,
   FSRef *newRef,
   FSSpec *newSpec
);
```

Parameters

parentRef
A pointer to the directory in which the resource file is to be created.

nameLength
The number of Unicode characters in the file's name.

name
A pointer to the Unicode name of the new resource file.

whichInfo
The catalog information fields to set. See the File Manager documentation for a description of the `FSCatalogInfoBitmap` data type.

catalogInfo
A pointer to the values for the catalog information fields. This pointer may be set to `NULL`. See the File Manager documentation for a description of the `FSCatalogInfo` data type.

forkNameLength
The number of Unicode characters in the fork's name.

forkName
A pointer to the Unicode name of the fork to initialize. If you pass `NULL` in this parameter, the data fork is used.

newRef
A pointer to a variable allocated by the caller, or `NULL`. On return, the new resource file.

newSpec
A pointer to a variable allocated by the caller, or `NULL`. On return, the new resource file.

Return Value
A result code. See "Resource Manager Result Codes" (page 1711).

Discussion
This function creates a new file and initializes the specified fork for storing resource data. If you don't specify the fork name, the data fork is used. This function makes it possible to store resources in the data fork of a file.

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

Declared In
`Resources.h`

FSCreateResourceFork

Creates a named fork for storing resource data.

```
OSErr FSCreateResourceFork (
   const FSRef *ref,
   UniCharCount forkNameLength,
   const UniChar *forkName,
   UInt32 flags
);
```

Parameters
ref
A pointer to the file to which to add the fork.

forkNameLength
The number of Unicode characters in the fork's name.

forkName
A pointer to the Unicode name of the fork to initialize. If you pass `NULL` in this parameter, the data fork is used.

flags
A value of type `UInt32`. You should pass `0`.

Return Value
A result code. See "Resource Manager Result Codes" (page 1711).

Discussion
This function creates the specified fork in an existing file and initializes the fork for storing resources. If the named fork already exists, this function does nothing and returns `errFSForkExists`. If you don't specify the fork name, the data fork is used. This function makes it possible to store resources in the data fork of a file.

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

Declared In
`Resources.h`

FSOpenOrphanResFile

Opens a resource file that is persistent across all contexts.

```
OSErr FSOpenOrphanResFile (
    const FSRef *ref,
    SignedByte permission,
    ResFileRefNum *refNum
);
```

Parameters

ref

A pointer to the resource file to open.

permission

A constant indicating the type of access with which to open the resource fork. For a description of the types of access you can request, see File Access Permission Constants in *File Manager Reference*.

refNum

A pointer to a variable allocated by the caller. On return, the reference number for accessing the open fork.

Return Value
A result code. See "Resource Manager Result Codes" (page 1711).

Discussion
This function loads a map and all preloaded resources into the system context and detaches the specified file from the context in which it was opened. If the file is already in the resource chain and a new instance is not opened, this function returns `paramErr`. Use this function with care, as it may fail if the map is very large or many resources are preloaded.

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

Declared In
`Resources.h`

FSOpenResFile

Opens the resource fork in a file specified with an `FSRef` structure.

```
ResFileRefNum FSOpenResFile (
    const FSRef *ref,
    SInt8 permission
);
```

Discussion
This function is not recommended. You should use a file's data fork instead of its resource fork to store resource data.

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

Declared In
`Resources.h`

FSOpenResourceFile

Opens a named fork in an existing resource file.

```
OSErr FSOpenResourceFile (
   const FSRef *ref,
   UniCharCount forkNameLength,
   const UniChar *forkName,
   SInt8 permissions,
   ResFileRefNum *refNum
);
```

Parameters

ref
A pointer to the file containing the fork to open.

forkNameLength
The number of Unicode characters in the fork's name.

forkName
A pointer to the Unicode name of the fork to open. If you pass `NULL` in this parameter, the data fork is used.

permissions
A constant indicating the type of access with which to open the fork. For a description of the types of access you can request, see File Access Permission Constants in *File Manager Reference*.

refNum
A pointer to a variable allocated by the caller. On return, the reference number for accessing the open fork.

Return Value
A result code. See "Resource Manager Result Codes" (page 1711).

Discussion
This function allows any named fork of a file to be used for storing resources. Passing in a null fork name will result in the data fork being used. You should use a file's data fork to store resource data.

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

Declared In
`Resources.h`

FSpCreateResFile

Creates an empty resource fork in a new or existing file. (Deprecated in Mac OS X v10.5. Use `FSCreateResourceFile` (page 1667) instead.)

```
void FSpCreateResFile (
   const FSSpec *spec,
   OSType creator,
   OSType fileType,
   ScriptCode scriptTag
);
```

Discussion
This function is not recommended. You should use a file's data fork instead of its resource fork to store resource data.

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
`Resources.h`

FSpOpenOrphanResFile

Opens a resource file that is persistent across all contexts. (Deprecated in Mac OS X v10.5. Use `FSOpenOrphanResFile` (page 1669) instead.)

```
OSErr FSpOpenOrphanResFile (
   const FSSpec *spec,
   SignedByte permission,
   ResFileRefNum *refNum
);
```

Return Value
A result code. See "Resource Manager Result Codes" (page 1711).

Discussion
`FSpOpenOrphanResFile` should be used to open a resource file that is persistent across all contexts. `FSpOpenOrphanResFile` loads everything into the system context and detaches the file from the context in which it was opened. If the file is already in the resource chain and a new instance is not opened, `FSpOpenOrphanResFile` will return a `paramErr`. Use this function with care, as it can and will fail if the map is very large or a lot of resources are preloaded.

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
`Resources.h`

FSpOpenResFile

Opens the resource fork in a file specified with an `FSSpec` structure. (Deprecated in Mac OS X v10.5. Use `FSOpenResourceFile` (page 1670) instead.)

```
ResFileRefNum FSpOpenResFile (
   const FSSpec *spec,
   SignedByte permission
);
```

Parameters

spec

A pointer to a file system specification record specifying the name and location of the file whose resource fork is to be opened. This function also makes the specified file the current resource file.

permission

A constant indicating the type of access with which to open the resource fork. For a description of the types of access you can request, see File Access Permission Constants in *File Manager Reference*.

Return Value

The file reference number for the resource fork. If the file reference number returned is greater than 0, you can use this number to refer to the resource fork in some other Resource Manager functions.

If you attempt to use this function to open a resource fork that is already open, it returns the existing file reference number or a new one, depending on the access permission for the existing access path. For example, your application receives a new file reference number after a successful request for read-only access to a file previously opened with write access, whereas it receives the same file reference number in response to a second request for write access to the same file. In this case, the function doesn't make that file the current resource file.

If the function fails to open the specified file's resource fork (for instance, because there's no file with the given file system specification record or because there are permission problems), it returns –1 as the file reference number. Use the `ResError` (page 1692) function to determine what kind of error occurred.

If an application attempts to open a second access path with write access and the application is different from the one that originally opened the resource fork, `FSpOpenResFile` returns –1, and the `ResError` function returns the result code `opWrErr`.

Discussion

This function is available only in System 7 and later versions of system software. You can determine whether `FSpOpenResFile` is available by calling the `Gestalt` function with the `gestaltFSAttr` selector code. If this function is not available to your application, you can use `HOpenResFile`, `OpenRFPerm`, or `OpenResFile` instead. The `HOpenResFile` (page 1686) function is preferred if `FSpOpenResFile` is not available. The `OpenRFPerm` (page 1689) function is an earlier version of `HOpenResFile` that is still supported but is more restricted in its capabilities.

The Resource Manager reads the resource map from the specified file's resource fork into memory. It also reads into memory every resource in the resource fork whose `resPreload` attribute is set.

You don't have to call this function to open the System file's resource fork or an application file's resource fork. These resource forks are opened automatically when the system and the application start up, respectively. To get the file reference number for your application, call the `CurResFile` (page 1664) function after your application starts up and before you open any other resource forks.

The `FSpOpenResFile` function checks that the information in the resource map is internally consistent. If it isn't, `ResError` returns the result code `mapReadErr`.

It's possible to create multiple, unique, read-only access paths to a resource fork using this function however, you should avoid doing so. If a resource fork is opened twice—once with read/write permission and once with read-only permission—two copies of the resource map exist in memory. If you change one of the resources in memory using one of the resource maps, the two resource maps become inconsistent and the file will appear damaged to the second resource map.

If you must use this technique for read-only access, call this function immediately before your application reads information from the file and close the file immediately afterward. Otherwise, your application may get unexpected results.

If you want to open the resource fork for another application (or any resource fork other than your application's that includes `'CODE'` resources), you must bracket your calls to this function with calls to the `SetResLoad` (page 1695) function with the `load` parameter set to `FALSE` and then to `TRUE`. You must also avoid making intersegment calls while the other application's resource fork is open. If you don't do this, the Segment Loader Manager treats any preloaded `'CODE'` resources as your code resources when you make an intersegment call that triggers a call to the `LoadSeg` function while the other application is first in the resource chain. In this case, your application can begin executing the other application's code, and severe problems will ensue. If you need to get `'CODE'` resources from the other application's resource fork, you can still prevent the Segment Loader Manager problem by calling the `UseResFile` (page 1700) function with your application's file reference number to make your application the current resource file.

To open a resource fork just for block-level operations, such as copying files without reading the resource map into memory, use the File Manager function `OpenRF`.

Special Considerations

Because there is no support for locking and unlocking file ranges on local disks in Mac OS X, regardless of whether File Sharing is enabled, you cannot open more than one path to a resource fork with read/ write permission. If you try to open a more than one path to a file's resource fork with `fsRdWrShPerm` permission, only the first attempt will succeed. Subsequent attempts will return an invalid reference number and the `ResError` function will return the error `opWrErr`.

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

Simple DrawSprocket

Declared In

`Resources.h`

FSpResourceFileAlreadyOpen

Checks whether a resource file is open. (Deprecated in Mac OS X v10.5. Use `FSResourceFileAlreadyOpen` (page 1674) instead.)

```
Boolean FSpResourceFileAlreadyOpen (
   const FSSpec *resourceFile,
   Boolean *inChain,
   ResFileRefNum *refNum
);
```

Parameters

resourceFile

The resource file to check.

inChain

A pointer to a variable allocated by the caller. On return, `true` if the resource file is in the resource chain, `false` otherwise.

refNum

A pointer to a variable allocated by the caller. On return, the reference number of the file if it is open.

Return Value

This function returns `true` if the resource file is already open and known by the Resource Manager—for example, it is either in the current resource chain or it is a detached resource 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

`Resources.h`

FSResourceFileAlreadyOpen

Checks whether a resource file is open.

```
Boolean FSResourceFileAlreadyOpen (
   const FSRef *resourceFileRef,
   Boolean *inChain,
   ResFileRefNum *refNum
);
```

Parameters

resourceFileRef

The resource file to check.

inChain

A pointer to a variable allocated by the caller. On return, `true` if the resource file is in the resource chain, `false` otherwise.

refNum

A pointer to a variable allocated by the caller. On return, the reference number of the file if it is open.

Return Value

This function returns `true` if the resource file is already open and known by the Resource Manager—for example, it is either in the current resource chain or it is a detached resource file.

Availability

Available in Mac OS X v10.0 and later.

Declared In
Resources.h

Get1IndResource

Returns a handle to a resource of a given type in the current resource file.

```
Handle Get1IndResource (
   ResType theType,
   ResourceIndex index
);
```

Parameters

theType

A resource type.

index

An integer ranging from 1 to the number of resources of a given type returned by the Count1Resources (page 1663) function, which is the number of resources of that type in the current resource file.

Return Value
A handle to a resource of the given type. If you call Get1IndResource repeatedly over the entire range of the index, it returns handles to all resources of the given type in the current resource file. If you provide an index that is either 0 or negative, the function returns NULL, and the ResError (page 1692) function returns the result code resNotFound. If the given index is larger than the value returned by Count1Resources (page 1663), Get1IndResource (page 1675) returns NULL, and ResError (page 1692) returns the result code resNotFound. If the resource to be read won't fit into memory, the function returns NULL, and ResError (page 1692) returns the appropriate result code.

Discussion
The function reads the resource data into memory if it's not already there, unless you've called the SetResLoad (page 1695) function with the load parameter set to FALSE. If you've called SetResLoad with the load parameter set to FALSE and the data isn't already in memory, Get1IndResource (page 1675) returns an empty handle (that is, a handle whose master pointer is set to NULL). This can also happen if you read resource data for a purgeable resource into memory and then call SetResLoad (page 1695) with the load parameter set to FALSE. If the resource data is later purged and you call the Get1IndResource (page 1675) function, the function returns an empty handle. You should test for an empty handle in these situations. To make the handle a valid handle to resource data in memory, you can call the LoadResource (page 1688) function.

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

Declared In
Resources.h

Get1IndType

Gets a resource types available in the current resource file.

```
void Get1IndType (
   ResType *theType,
   ResourceIndex index
);
```

Parameters

theType

On return, the resource type with the specified index in the current resource file.

You can call this function repeatedly over the entire range of the index to get all the resource types available in the current resource file. If the given index isn't in the range from 1 to the number of resource types as returned by `Count1Types`, this parameter contains four null characters (ASCII code 0).

index

An integer ranging from 1 to the number of resource types in the current resource file, as returned by the `Count1Types` (page 1663) function.

Discussion
To check for errors, call the `ResError` (page 1692) function.

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

Declared In
`Resources.h`

Get1NamedResource

Gets a named resource in the current resource file.

```
Handle Get1NamedResource (
   ResType theType,
   ConstStr255Param name
);
```

Parameters

theType

The resource type of the resource about which you wish to retrieve data.

name

A name that uniquely identifies the resource about which you wish to retrieve data.

Return Value
If the function finds an entry for the resource in the current resource file's resource map and the entry contains a valid handle, the function returns that handle. If the entry contains a handle whose value is `NULL`, and if you haven't called the `SetResLoad` (page 1695) function with the `load` parameter set to `FALSE`, `Get1NamedResource` attempts to read the resource into memory. If it can't find the resource data, the function returns `NULL`, and the `ResError` (page 1692) function returns the result code `resNotFound`. The `Get1NamedResource` function also returns `NULL` if the resource data to be read into memory won't fit, in which case `ResError` returns an appropriate Memory Manager result code.

If you call this function with a resource type that can't be found in the resource map of the current resource file, the function returns `NULL`, but `ResError` returns the result code `noErr`. You should always check that the value of the returned handle is not `NULL`.

Discussion
The function searches the current resource file's resource map in memory for the specified resource. You can change the search order by calling the `UseResFile` (page 1700) function before `Get1NamedResource`.

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

Declared In
`Resources.h`

Get1Resource

Gets resource data for a resource in the current resource file.

```
Handle Get1Resource (
   ResType theType,
   ResID theID
);
```

Parameters

theType
The resource type of the resource about which you wish to retrieve data.

theID
An integer that uniquely identifies the resource about which you wish to retrieve data.

Return Value
If the function finds an entry for the resource in the current resource file's resource map and the entry contains a valid handle, it returns that handle. If the entry contains a handle whose value is `NULL`, and if you haven't called the `SetResLoad` (page 1695) function with the `load` parameter set to `FALSE`, `Get1Resource` attempts to read the resource into memory.

If the function can't find the resource data, it returns `NULL`, and `ResError` returns the result code `resNotFound`. The `Get1Resource` function also returns `NULL` if the resource data to be read into memory won't fit, in which case `ResError` returns an appropriate Memory Manager result code.

If you call this function with a resource type that can't be found in the resource map of the current resource file, the function returns `NULL`, but `ResError` returns the result code `noErr`. You should always check that the value of the returned handle is not `NULL`.

Discussion
The function searches the current resource file's resource map in memory for the specified resource.

You can change the resource map search order by calling the `UseResFile` (page 1700) function before `Get1Resource`.

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

Related Sample Code
QTMetaData

Declared In
`Resources.h`

GetIndResource

Returns a handle to a resource of a given type in resource forks open to your application.

```
Handle GetIndResource (
   ResType theType,
   ResourceIndex index
);
```

Parameters

theType

A resource type.

index

An integer ranging from 1 to the number of resources of a given type returned by the `CountResources` (page 1664) function, which is the number of resources of that type in all resource forks open to your application.

Return Value

A handle to a resource of the given type. If you call this function repeatedly over the entire range of the index, it returns handles to all resources of the given type in all resource forks open to your application. The function returns handles for all resources in the most recently opened resource fork first, and then for those in resource forks opened earlier in reverse chronological order. If you provide an index to that is either 0 or negative, the function returns `NULL`, and the `ResError` (page 1692) function returns the result code `resNotFound`. If the given index is larger than the value returned by `CountResources`, the function returns `NULL`, and `ResError` (page 1692) returns the result code `resNotFound`. If the resource to be read won't fit into memory, the function returns `NULL`, and `ResError` (page 1692) returns the appropriate result code.

Discussion

This function reads the resource data into memory if it's not already there, unless you've called the `SetResLoad` (page 1695) function with the `load` parameter set to `FALSE`.

If you've called `SetResLoad` (page 1695) with the `load` parameter set to `FALSE` and the data isn't already in memory, the function returns an empty handle (a handle whose master pointer is set to NULL). This can also happen if you read resource data for a purgeable resource into memory and then call `SetResLoad` with the `load` parameter set to `FALSE`. If the resource data is later purged and you call the `GetIndResource` function, the function returns an empty handle. You should test for an empty handle in these situations. To make the handle a valid handle to resource data in memory, you can call the `LoadResource` (page 1688) function.

The `UseResFile` (page 1700) function affects which file the Resource Manager searches first when looking for a particular resource; this is not the case when you use `GetIndResource` to get an indexed resource.

If you want to find out how many resources of a given type are in a particular resource fork, set the current resource file to that resource fork, then call the `Count1Resources` (page 1663) function and use the `Get1IndResource` (page 1675) function to get handles to the resources of that type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Resources.h`

GetIndType

Gets a resource type available in resource forks open to your application.

```
void GetIndType (
   ResType *theType,
   ResourceIndex index
);
```

Parameters

theType

On return, a pointer to the resource type for the specified index among all the resource forks open to your application.

You can call this function repeatedly over the entire range of the index to get all the resource types available in all resource forks open to your application. If the given index isn't in the range from 1 to the number of resource types as returned by `CountTypes`, this parameter contains four null characters (ASCII code 0).

index

An integer ranging from 1 to the number of resource types in all resource forks open to your application, as returned by `CountTypes` (page 1664) function.

Discussion

To check for errors, call the `ResError` (page 1692) function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Resources.h`

GetMaxResourceSize

Returns the approximate size of a resource.

```
long GetMaxResourceSize (
   Handle theResource
);
```

Parameters

theResource

A handle to the resource whose size you wish to retrieve.

Return Value

The approximate size, in bytes, of the resource. Unlike the `GetResourceSizeOnDisk` (page 1684) function, this function does not check the resource on disk instead, it either checks the resource size in memory or, if the resource is not in memory, calculates its size on the basis of information in the resource map in memory. This gives you an approximate size for the resource that you can count on as the resource's maximum size. It's possible that the resource is actually smaller than the offsets in the resource map indicate because the file has not yet been compacted. If you want the exact size of a resource on disk, either call `GetResourceSizeOnDisk` or call the `UpdateResFile` (page 1700) function before calling `GetMaxResourceSize`. If the handle isn't a handle to a valid resource, the function returns –1, and the `ResError` (page 1692) function returns the result code `resNotFound`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Resources.h`

GetNamedResource

Gets a named resource.

```
Handle GetNamedResource (
   ResType theType,
   ConstStr255Param name
);
```

Parameters

theType

The resource type of the resource about which you wish to retrieve data.

name

A name that uniquely identifies the resource about which you wish to retrieve data. Strings passed in this parameter are case-sensitive.

Return Value

If the function finds the specified resource entry in one of the resource maps and the entry contains a valid handle, the function returns that handle. If the entry contains a handle whose value is `NULL`, and if you haven't called the `SetResLoad` (page 1695) function with the `load` parameter set to `FALSE`, `GetNamedResource` attempts to read the resource into memory.

If the function can't find the resource data, it returns `NULL`, and the `ResError` (page 1692) function returns the result code `resNotFound`. The function also returns `NULL` if the resource data to be read into memory won't fit, in which case `ResError` returns an appropriate Memory Manager result code. If you call `GetNamedResource` with a resource type that can't be found in any of the resource maps of the open resource forks, the function returns `NULL` as well, but `ResError` returns the result code `noErr`. You should always check that the value of the returned handle is not `NULL`.

Discussion

The function searches the resource maps in memory for the specified resource. The resource maps in memory, which represent all the open resource forks, are arranged as a linked list. When the function searches this list, it starts with the current resource file and progresses through the list in order (that is, in reverse chronological order in which the resource forks were opened) until it finds the resource's entry in one of the resource maps.

You can change the resource map search order by calling the `UseResFile` (page 1700) function before `GetNamedResource`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Resources.h`

GetNextFOND

Gets the next FOND handle.

```
Handle GetNextFOND (
   Handle fondHandle
);
```

Availability

Available in Mac OS X v10.0 and later.

Declared In
Resources.h

GetNextResourceFile

Retrieves the next resource file in the resource chain.

```
OSErr GetNextResourceFile (
   ResFileRefNum curRefNum,
   ResFileRefNum *nextRefNum
);
```

Parameters

curRefNum

A value of type SInt16 representing the current reference number of a resource file.

nextRefNum

A pointer to a value of type SInt16. On return, this points to the next resource file in the resource chain.

Return Value
A result code. See "Resource Manager Result Codes" (page 1711).

Discussion
`GetNextResourceFile` can be used to iterate over resource files in the resource chain. By passing a valid reference number in the `curRefNum` parameter, the function returns the reference number of the next file in the resource chain. If the resource file specified by the `curRefNum` parameter is not found in the resource chain, the `GetNextResourceFile` function returns the error code `resFNotFound`. When the end of the chain is reached `GetNextResourceFile` returns `noErr` and the value of the `nextRefNum` parameter is `NIL`.

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

Declared In
Resources.h

GetResAttrs

Gets a resource's attributes.

```
ResAttributes GetResAttrs (
   Handle theResource
);
```

Parameters

theResource

A handle to the resource whose attributes you wish to retrieve. If the value of this parameter isn't a handle to a valid resource, the function does nothing, and the `ResError` (page 1692) function returns the result code `resNotFound`.

Return Value
The resource's attributes as recorded in its entry in the resource map in memory. The function returns the resource's attributes in the low-order byte of the function result. Each attribute is identified by a specific bit in the low-order byte. If the bit corresponding to an attribute contains 1, then that attribute is set if the bit contains 0, then that attribute is not set.

Discussion
To change a resource's attributes in the resource map in memory, use the `SetResAttrs` (page 1693) function.

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

Declared In
`Resources.h`

GetResFileAttrs

Gets the attributes of a resource fork.

```
ResFileAttributes GetResFileAttrs (
   ResFileRefNum refNum
);
```

Parameters

refNum

A file reference number for the resource fork whose attributes you want to get. Specify 0 in this parameter to get the attributes of the System file's resource fork.

Return Value
The attributes of the file's resource fork. If there's no open resource fork for the given file reference number, the function does nothing, and the `ResError` (page 1692) function returns the result code `resFNotFound`. Like individual resources, resource forks have attributes that are specified by bits in the low-order byte of a word.

Discussion
The Resource Manager sets the `mapChanged` attribute for the resource fork when you call the `ChangedResource` (page 1661), the `AddResource` (page 1660), or the `RemoveResource` (page 1692) function.

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

Declared In
`Resources.h`

GetResInfo

Gets a resource's resource ID, resource type, and resource name.

```
void GetResInfo (
   Handle theResource,
   ResID *theID,
   ResType *theType,
   Str255 name
);
```

Parameters

theResource

A handle to the resource for which you want to retrieve information. If the handle isn't a valid handle to a resource, the function does nothing to determine whether this has occurred, call the `ResError` (page 1692) function.

theID
On return, a pointer to the resource ID of the specified resource.

theType
On return, a pointer to the resource type of the specified resource.

name
On return, the name of the specified resource.

Discussion
To set a resource's ID, resource type, or resource name, use the `SetResInfo` (page 1694) function.

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

Declared In
`Resources.h`

GetResource

Gets resource data for a resource specified by resource type and resource ID.

```
Handle GetResource (
   ResType theType,
   ResID theID
);
```

Parameters

theType
The resource type of the resource about which you wish to retrieve data.

theID
An integer that uniquely identifies the resource about which you wish to retrieve data.

Return Value
If the function finds the specified resource entry in one of the resource maps and the entry contains a valid handle, it returns that handle. If the entry contains a a handle whose value is `NULL`, and if you haven't called the `SetResLoad` (page 1695) function with the `load` parameter set to `FALSE`, `GetResource` attempts to read the resource into memory.

If the function can't find the resource data, it returns `NULL`, and the `ResError` (page 1692) function returns the result code `resNotFound`. The `GetResource` function also returns `NULL` if the resource data to be read into memory won't fit, in which case `ResError` returns an appropriate Memory Manager result code. If you call `GetResource` with a resource type that can't be found in any of the resource maps of the open resource forks, the function returns `NULL`, but `ResError` returns the result code `noErr`. You should always check that the value of the returned handle is not `NULL`.

Discussion
The function searches the resource maps in memory for the specified resource. The resource maps in memory, which represent all the open resource forks, are arranged as a linked list. When searching this list, the function starts with the current resource file and progresses through the list (that is, searching the resource maps in reverse order of opening) until it finds the resource's entry in one of the resource maps.

Before reading the resource data into memory, the Resource Manager calls the Memory Manager to allocate a relocatable block for the resource data. The Memory Manager allocates the block, assigns a master pointer to the block, and returns to the Resource Manager a pointer to the master pointer. The Resource Manager then installs this handle in the resource map.

You can change the resource map search order by calling the `UseResFile` (page 1700) function before calling `GetResource`.

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

Related Sample Code
Simple DrawSprocket

Declared In
`Resources.h`

GetResourceSizeOnDisk

Returns the exact size of a resource.

```
long GetResourceSizeOnDisk (
   Handle theResource
);
```

Parameters

`theResource`
A handle to the resource whose size you wish to retrieve.

Return Value
The exact size, in bytes, of the resource. If the handle isn't a handle to a valid resource, the function returns –1, and the `ResError` (page 1692) function returns the result code `resNotFound`.

Discussion
This function checks the resource on disk, not in memory. You can call this function before reading a resource into memory to make sure there's enough memory available to do so successfully.

The `GetResourceSizeOnDisk` function is also available as the `SizeResource` function.

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

Declared In
`Resources.h`

GetTopResourceFile

Retrieves the topmost resource file in the current resource chain.

```
OSErr GetTopResourceFile (
   ResFileRefNum *refNum
);
```

Parameters

`refNum`
A pointer to a value of type `SInt16`. On return, this points to the top most resource file in the current resource chain.

Return Value
A result code. See "Resource Manager Result Codes" (page 1711). If the resource chain is empty, `resFNotFound` is returned.

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

Declared In
`Resources.h`

HCreateResFile

Creates an empty resource fork, when the `FSpCreateResFile` function is not available. (Deprecated in Mac OS X v10.5. Use `FSCreateResourceFile` (page 1667) instead.)

```
void HCreateResFile (
   FSVolumeRefNum vRefNum,
   long dirID,
   ConstStr255Param fileName
);
```

Discussion
This function is not recommended. You should use a file's data fork instead of its resource fork to store resource data.

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
`Resources.h`

HomeResFile

Gets the file reference number associated with a particular resource.

```
ResFileRefNum HomeResFile (
   Handle theResource
);
```

Parameters
`theResource`
A handle to the resource for which you wish to get the associated file reference number.

Return Value
The file reference number for the resource fork containing the specified resource. If the given handle isn't a handle to a resource, the function returns –1, and the `ResError` (page 1692) function returns the result code `resNotFound`. If `HomeResFile` returns 0, the resource is in the System file's resource fork. If it returns 1, the resource is ROM-resident.

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

Declared In
Resources.h

HOpenResFile

Opens a file's resource fork, when the FSpOpenResFile function is not available. (Deprecated in Mac OS X v10.5. Use FSOpenResourceFile (page 1670) instead.)

```
ResFileRefNum HOpenResFile (
   FSVolumeRefNum vRefNum,
   long dirID,
   ConstStr255Param fileName,
   SInt8 permission
);
```

Parameters

vRefNum
The volume reference number of the volume on which the file is located.

dirID
The directory ID of the directory where the file is located.

fileName
The name of the file whose resource fork is to be opened.

permission
A constant indicating the type of access with which to open the resource fork. For a description of the types of access you can request, see File Access Permission Constants in *File Manager Reference*.

Return Value
The file reference number for the file. You can use this file reference number to refer to the file in other Resource Manager functions. The function also makes this file the current resource file. If the file's resource fork is already open, the function returns the file reference number but does not make that file the current resource file. If the function fails to open the specified file's resource fork (because there's no file with the specified name or because there are permission problems), it returns –1 as the file reference number. Use the ResError (page 1692) function to determine what kind of error occurred.

Versions of system software before System 7 do not allow you to use this function to open a second access path, with write access, to a resource fork. In this case, the function returns the reference number already assigned to the file.

Discussion
The Resource Manager reads the resource map from the resource fork of the specified file into memory. It also reads into memory every resource whose resPreload attribute is set.

You don't have to call this function to open the System file's resource fork or an application file's resource fork. These files are opened automatically when the system and the application start up, respectively. To get the file reference number for your application, call the CurResFile (page 1664) function after the application starts up and before you open the resource forks for any other files.

The HOpenResFile function checks that the information in the resource map is internally consistent. If it isn't, ResError returns the result code mapReadErr. It's possible to create multiple, unique, read-only access paths to a resource fork using HOpenResFile; however, you should avoid doing so, to prevent inconsistencies between multiple copies of the resource map. See the discussion of this issue in relation to FSpOpenResFile (page 1671). The HOpenResFile function works the same way.

To open a resource fork just for block-level operations, such as copying files without reading the resource map into memory, use the File Manager function `OpenRF`.

If you want to open the resource fork for another application (or any resource fork other than your application's that includes `'CODE'` resources), you must bracket your calls to `HOpenResFile` with calls to the `SetResLoad` (page 1695) function with the `load` parameter set to `FALSE` and then to `TRUE`. You must also avoid making intersegment calls while the other application's resource fork is open. The discussion of this issue in relation to `FSpOpenResFile` (page 1671) also applies to `HOpenResFile`.

Special Considerations

Because there is no support for locking and unlocking file ranges on local disks in Mac OS X, regardless of whether File Sharing is enabled, you cannot open more than one path to a resource fork with read/write permission. If you try to open a more than one path to a file's resource fork with `fsRdWrShPerm` permission, only the first attempt will succeed. Subsequent attempts will return an invalid reference number and the `ResError` function will return the error `opWrErr`.

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
`Resources.h`

InsertResourceFile

Inserts a resource file into the current resource chain at the specified location.

```
OSErr InsertResourceFile (
   ResFileRefNum refNum,
   RsrcChainLocation where
);
```

Parameters

refNum
A value of type SInt16 indicating the reference number of the resource file to insert into the resource chain.

where
A value of type RsrcChainLocation indicating where in the resource chain the resource file should be inserted. See the `RsrcChainLocation` data type.

Return Value
A result code. See "Resource Manager Result Codes" (page 1711).

Discussion
If the file is already in the resource chain, it is removed and reinserted at the specified location. If the file has been detached, it is added to the resource chain at the specified location Returns resFNotFound if it's not currently open.

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

Declared In
`Resources.h`

InvokeResErrUPP

Calls your callback function.

```
void InvokeResErrUPP (
    OSErr thErr,
    ResErrUPP userUPP
);
```

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

Declared In
`Resources.h`

LoadResource

Gets resource data after you've called the `SetResLoad` function with the `load` parameter set to `FALSE` or when the resource is purgeable.

```
void LoadResource (
    Handle theResource
);
```

Parameters

`theResource`

A handle to a resource. Given this handle, the function reads the resource data into memory. If the resource is already in memory, or if the this parameter doesn't contain a handle to a resource, then the function does nothing. To determine whether either of these situations occurred, call the `ResError` (page 1692) function. If the resource is already in memory, `ResError` returns `noErr`; if the handle is not a handle to a resource, `ResError` returns `resNotFound`.

Discussion
If you've changed the resource data for a purgeable resource and the resource is purged before being written to the file, the changes will be lost. In this case, this function rereads the original resource from the file's resource fork. You should use the `ChangedResource` (page 1661) or `SetResPurge` (page 1697) function before calling `LoadResource` to ensure that changes made to purgeable resources are written to the resource fork.

Availability
Available in Mac OS X 10.0 and later.

Declared In
`Resources.h`

NewResErrUPP

Creates a new universal procedure pointer (UPP) to your callback function.

```
ResErrUPP NewResErrUPP (
    ResErrProcPtr userRoutine
);
```

Return Value
See `ResErrUPP` (page 1704) for more information.

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

Declared In
`Resources.h`

OpenRFPerm

Opens a file's resource fork, when the `FSpOpenResFile` and `HOpenResFile` functions are not available. (Deprecated in Mac OS X v10.5. Use `FSOpenResourceFile` (page 1670) instead.)

```
ResFileRefNum OpenRFPerm (
   ConstStr255Param fileName,
   FSVolumeRefNum vRefNum,
   SInt8 permission
);
```

Parameters

`fileName`
The name of the file whose resource fork is to be opened.

`vRefNum`
The volume reference number or directory ID for the volume or directory in which the file is located.

`permission`
A constant indicating the type of access with which to open the resource fork. For a description of the types of access you can request, see File Access Permission Constants in *File Manager Reference*.

Return Value
The file reference number for the file whose resource fork it has opened. You can use this file reference number to refer to the file in other Resource Manager functions. The function also makes this file the current resource file. If the file's resource fork is already open, the function returns the file reference number but does not make that file the current resource file.

If the function fails to open the specified file's resource fork (because there's no file with the given name or because there are permission problems), it returns –1 as the file reference number. Use the `ResError` (page 1692) function to determine what kind of error occurred.

Versions of system software before System 7 do not allow you to use this function to open a second access path, with write access, to a resource fork. In this case, the function returns the reference number already assigned to the file.

Discussion
You can use this function if the `FSpOpenResFile` (page 1671) function is not available. You can determine whether `FSpOpenResFile` is available by calling the `Gestalt` function with the `gestaltFSAttr` selector code. The `HOpenResFile` function allows you to specify both a directory ID and a volume reference number, and is therefore preferred if `FSpOpenResFile` is not available. The `OpenRFPerm` is an earlier versions of `HOpenResFile` that is still supported but is more restricted in its capabilities.

The Resource Manager reads the resource map from the resource fork for the specified file into memory. It also reads into memory every resource in the resource fork whose `resPreload` attribute is set.

You don't have to call this function to open the System file's resource fork or an application file's resource fork. These files are opened automatically when the system and the application start up, respectively. To get the file reference number for your application, call the `CurResFile` (page 1664) function after the application starts up and before you open the resource forks for any other files.

This function checks that the information in the resource map is internally consistent. If it isn't, `ResError` returns the result code `mapReadErr`. It's possible to create multiple, unique, read-only access paths to a resource fork using this function however, you should avoid doing so, to prevent inconsistencies between multiple copies of the resource map.

To open a resource fork just for block-level operations, such as copying files without reading the resource map into memory, use the File Manager function `OpenRF`.

If you want to open the resource fork for another application (or any resource fork other than your application's that includes `'CODE'` resources), you must bracket your calls to this function with calls to the `SetResLoad` (page 1695) function with the `load` parameter set to `FALSE` and then to `TRUE`. You must also avoid making intersegment calls while the other application's resource fork is open.

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
`Resources.h`

ReadPartialResource

Reads part of a resource into memory and work with a small subsection of a large resource.

```
void ReadPartialResource (
   Handle theResource,
   long offset,
   void *buffer,
   long count
);
```

Parameters

theResource
A handle to the resource you wish to read.

offset
The beginning of the resource subsection to be read, measured in bytes from the beginning of the resource.

buffer
A pointer to the buffer into which the partial resource is to be read. Your application is responsible for the buffer's memory management. You cannot use the `ReleaseResource` (page 1691) function to release the memory the buffer occupies.

count
The length of the resource subsection.

Discussion
This function always tries to read resources from disk. If a resource is already in memory, the Resource Manager still reads it from disk, and the `ResError` (page 1692) function returns the result code `resourceInMemory`. If you try to read past the end of a resource or the value of the `offset` parameter is out of bounds, `ResError` returns the result code `inputOutOfBounds`. If the handle in the parameter `theResource` doesn't refer to a resource in an open resource fork, `ResError` returns the result code `resNotFound`.

You may experience problems if you have a copy of a resource in memory when you are using the partial resource functions. If you have modified the copy in memory and then access the resource on disk using this function, the function reads the data on disk, not the data in memory, which is referenced through the resource's handle.

When using partial resource functions, you should call the `SetResLoad` (page 1695) function, specifying `FALSE` for the `load` parameter, before you call `GetResource`. Using the `SetResLoad` function prevents the Resource Manager from reading the entire resource into memory. Be sure to restore the normal state by calling `SetResLoad` again, with the `load` parameter set to `TRUE`, immediately after you call the `GetResource` (page 1683) function. Then use `ReadPartialResource` to read a portion of the resource into a buffer.

If the entire resource is in memory and you want only part of its data, it's faster to use the Memory Manager function `BlockMove` instead of the `ReadPartialResource` function. If you read a partial resource into memory and then change its size, you can use the `SetResourceSize` (page 1696) function to change the entire resource's size on disk as necessary

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

Declared In
`Resources.h`

ReleaseResource

Releases the memory a resource occupies when you have finished using it.

```
void ReleaseResource (
   Handle theResource
);
```

Parameters

`theResource`

A handle to the resource which you wish to release. The function sets the master pointer of the resource's handle in the resource map in memory to `NULL`. If your application previously obtained a handle to that resource, the handle is no longer valid. If your application subsequently calls the Resource Manager to get the released resource, the Resource Manager assigns a new handle.

If the given resource isn't a handle to a resource, the function does nothing, and the `ResError` (page 1692) function returns the result code `resNotFound`. Be aware that `ReleaseResource` won't release a resource whose `resChanged` attribute has been set, but `ResError` still returns the result code `noErr`.

Special Considerations

Do not use this function to release a System resource that might be shared by several applications.

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

Related Sample Code
Simple DrawSprocket

Declared In
`Resources.h`

RemoveResource

Removes a resource's entry from the current resource file's resource map in memory.

```
void RemoveResource (
   Handle theResource
);
```

Parameters

theResource

A handle to the resource which you wish to detach. If the `resProtected` attribute for the resource is set or if this parameter doesn't contain a handle to a resource, the function does nothing, and the `ResError` (page 1692) function returns the result code `rmvResFailed`.

Discussion

The `RemoveResource` function does not dispose of the handle you pass into it; to do so you must call the Memory Manager function `DisposeHandle` after calling `RemoveResource`. You should dispose the handle if you want to release the memory before updating or closing the resource fork.

If you've removed a resource, the Resource Manager writes the entire resource map when it updates the resource fork, and all changes made to the resource map become permanent. If you want any of the changes to be temporary, you should restore the original information before the Resource Manager updates the resource fork.

The `RemoveResource` function is also available as the `RmveResource` function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Resources.h`

ResError

Determines what error occurred, if any, after calling a Resource Manager function.

```
OSErr ResError (
   void
);
```

Return Value

A result code. See "Resource Manager Result Codes" (page 1711). If no error occurred, the function returns `noErr`. If an error occurs at the Resource Manager level, the function returns one of the result codes specific to the Resource Manager. If an error occurs at the Operating System level, the function returns an Operating System result code. In certain cases, the `ResError` function returns `noErr` even though a Resource Manager function was unable to perform the requested operation. See the individual function descriptions for details about the circumstances under which this happens.

Discussion

Resource Manager functions do not report error information directly. Instead, after calling a Resource Manager function, your application should call this function to determine whether an error occurred. You also can use this function to check for an error after application startup (system software opens the resource fork of your application during application startup).

Resource Manager functions usually return NULL or –1 as the function result when there's an error. For Resource Manager functions that return –1, your application can call the ResError function to determine the specific error that occurred. For Resource Manager functions that return handles, your application should always check whether the value of the returned handle is NULL. If it is, your application can use this function to obtain specific information about the nature of the error. Note, however, that in some cases ResError returns noErr even though the value of the returned handle is NULL.

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

Related Sample Code
QTMetaData
Simple DrawSprocket

Declared In
Resources.h

SetResAttrs

Sets a resource's attributes in the resource map in memory.

```
void SetResAttrs (
   Handle theResource,
   ResAttributes attrs
);
```

Parameters

theResource

A handle to the resource whose attributes you wish to set. If the value of this parameter isn't a valid handle to a resource, the function does nothing, and the ResError (page 1692) function returns the result code resNotFound.

attrs

The resource attributes to set. The resProtected attribute changes immediately. Other attribute changes take effect the next time the specified resource is read into memory but are not made permanent until the Resource Manager updates the resource fork.

Each attribute is identified by a specific bit in the low-order byte of a word. If the bit corresponding to an attribute contains 1, then that attribute is set; if the bit contains 0, then that attribute is not set.

Discussion
This function changes the information in the resource map in memory, not in the file on disk. If you want the Resource Manager to write the modified resource map to disk after a subsequent call to the UpdateResFile (page 1700) function or when your application terminates, call the ChangedResource (page 1661) function after you call SetResAttrs.

Do not use this function to change a purgeable resource. If you make a purgeable resource nonpurgeable by setting the resPurgeable attribute with this function, the resource doesn't become nonpurgeable until the next time the specified resource is read into memory. Thus, the resource might be purged while you're changing it.

You can check for errors using the ResError function. SetResAttrs does not return an error if you are setting the attributes of a resource in a resource file that has a read-only resource map. To find out whether this is the case, use the GetResAttrs (page 1681) function.

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

Declared In
`Resources.h`

SetResFileAttrs

Sets a resource fork's attributes.

```
void SetResFileAttrs (
   ResFileRefNum refNum,
   ResFileAttributes attrs
);
```

Parameters

refNum

A file reference number for the resource fork whose attributes you want to set. If this value is 0, it represents the System file's resource fork. However, you shouldn't change the attributes of the System file's resource fork. If there's no resource fork with the given reference number, the function does nothing, and the `ResError` (page 1692) function returns the result code `noErr`.

attrs

The attributes to set. Like individual resources, resource forks have attributes that are specified by bits in the low-order byte of a word. When the Resource Manager first creates a resource fork after a call to `FSpOpenResFile` (page 1671) or a related function, it does not set any of the resource fork's attributes—that is, they are all set to 0.

Discussion
The Resource Manager sets the `mapChanged` attribute for the resource fork when you call the `ChangedResource` (page 1661), the `AddResource` (page 1660), or the `RemoveResource` (page 1692) function.

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

Declared In
`Resources.h`

SetResInfo

Sets the name and resource ID of a resource.

```
void SetResInfo (
   Handle theResource,
   ResID theID,
   ConstStr255Param name
);
```

Parameters

theResource

A handle to the resource whose name and ID you wish to set.

theID

The new resource ID. If the parameter `theResource` doesn't contain a handle to an existing resource, the function does nothing, and the `ResError` (page 1692) function returns the result code `resNotFound`.

name

The new name of the specified resource. If you pass an empty string for the `name` parameter, the resource name is not changed.

Discussion

The function changes the information in the resource map in memory, not in the resource file itself. Do not change a system resource's resource ID or name. Other applications may already access the resource and may not work properly if you change the resource ID, resource name, or both.

If the resource map becomes too large to fit in memory (for example, after an unnamed resource is given a name), this function does nothing, and `ResError` returns an appropriate Memory Manager result code. The same is true if the resource data in memory can't be written to the resource fork (for example, because the disk is full). If the `resProtected` attribute is set for the resource, `SetResInfo` does nothing, and `ResError` returns the result code `resAttrErr`.

If you want to write changes to the resource map on disk after updating the resource map in memory, call the `ChangedResource` (page 1661) function for the same resource after you call `SetResInfo`. Even if you don't call `ChangedResource` after using this function to change the name and resource ID of a resource, the change may be written to disk when the Resource Manager updates the resource fork. If you call `ChangedResource` for any resource in the same resource fork, or if you add or remove a resource, the Resource Manager writes the entire resource map to disk after a call to the `UpdateResFile` (page 1700) function or when your application terminates. In these cases, all changes to resource information in the resource map become permanent. If you want any of the changes to be temporary, you should restore the original information before the resource is updated.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Resources.h`

SetResLoad

Enables and disables automatic loading of resource data into memory for functions that return handles to resources.

```
void SetResLoad (
   Boolean load
);
```

Parameters

load

Determines whether Resource Manager functions should read resource data into memory. If you set this parameter to `TRUE`, Resource Manager functions that return handles will, during subsequent calls, automatically read resource data into memory if it is not already in memory; if you set this parameter to `FALSE`, Resource Manager functions will not automatically read resource data into memory. Instead, such functions return a handle whose master pointer is set to `NULL` unless the resource is already in memory. In addition, when first opening a resource fork the Resource Manager won't load into memory resources whose `resPreload` attribute is set. The default setting is `TRUE`.

If you call the function with this parameter set to `FALSE`, be sure to call `SetResLoad` with this parameter set to `TRUE` as soon as possible. Other parts of system software that call the Resource Manager expect this value to be `TRUE`, and some functions won't work if resources are not loaded automatically.

Discussion

You can use the `SetResLoad` function when you want to read from the resource map without reading the resource data into memory. To read the resource data into memory after a call to this function, call the `LoadResource` function.

To check for errors, call the `ResError` (page 1692) function.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Simple DrawSprocket

Declared In

`Resources.h`

SetResourceSize

Sets the size of a resource on disk.

```
void SetResourceSize (
   Handle theResource,
   long newSize
);
```

Parameters

theResource

A handle to the resource which you wish to change.

newSize

The size, in bytes, that you want the resource to occupy on disk. If the specified size is smaller than the resource's current size on disk, you lose any data from the cutoff point to the end of the resource. If the specified size is larger than the resource's current size on disk, all data is preserved, but the additional area is uninitialized (arbitrary data).

Discussion
This function is normally used only with the `ReadPartialResource` (page 1690) and `WritePartialResource` (page 1701) functions.

This function sets the size field of the specified resource on disk without writing the resource data. You can change the size of any resource, regardless of the amount of memory you have available.

If you read a partial resource into memory and then change its size, you must use this function to change the entire resource's size on disk as necessary. For example, suppose the entire resource occupies 1 MB and you use `ReadPartialResource` to read in a 200 KB portion of the resource. If you then increase the size of this partial resource to 250 KB, you must call `SetResourceSize` to set the size of the resource on disk to 1.05 MB. Note that in this case you must also keep track of the resource data on disk and move any data that follows the original partial resource on disk. Otherwise, there will be no space for the additional 50 KB when you call `WritePartialResource` to write the modified partial resource to disk.

Under certain circumstances, the Resource Manager overrides the size you set with a call to this function. For instance, suppose you read an entire resource into memory by calling `GetResource` (page 1683) or related functions, then use `SetResourceSize` successfully to set the resource size on disk, and finally attempt to write the resource to disk using the `UpdateResFile` (page 1700) or `WriteResource` (page 1702) functions. In this case, the Resource Manager adjusts the resource size on disk to conform with the size of the resource in memory.

If the disk is locked or full, or the file is locked, this function does nothing, and the `ResError` (page 1692) function returns an appropriate File Manager result code. If the resource is in memory, the Resource Manager tries to set the size of the resource on disk. If the attempt succeeds, `ResError` returns the result code `resourceInMemory`, and the Resource Manager does not update the copy in memory. If the attempt fails, `ResError` returns an appropriate File Manager result code.

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

Declared In
`Resources.h`

SetResPurge

Tells the Memory Manager to pass the handle of a resource to the Resource Manager before purging the data specified by that handle.

```
void SetResPurge (
   Boolean install
);
```

Parameters

install

Specifies whether the Memory Manager checks with the Resource Manager before purging a resource handle.

Specify `TRUE` to make the Memory Manager pass the handle for a resource to the Resource Manager before purging the resource data to which the handle points. The Resource Manager determines whether the handle points to a resource in the application heap. It also checks if the resource's `resChanged` attribute is set to 1. If these two conditions are met, the Resource Manager calls the `WriteResource` (page 1702) function to write the resource's resource data to the resource fork before returning control to the Memory Manager.

If you call this function with this parameter set to `TRUE` and then call the Memory Manager function `MoveHHi` to move a handle to a resource, the Resource Manager calls the `WriteResource` function to write the resource data to disk even if the data has not been changed. To prevent this, call `SetResPurge` with this parameter set to `FALSE` before you call `MoveHHi`, then call `SetResPurge` again with this parameter set to `TRUE` immediately after you call `MoveHHi`.

Whenever you call this function with this parameter set to `TRUE`, the Resource Manager installs its own purge-warning function, overriding any purge-warning function you've specified to the Memory Manager.

Specify `FALSE` to restore the normal state, so that the Memory Manager purges resource data when it needs to without calling the Resource Manager.

Discussion

You can use this function in applications that modify purgeable resources. You should also take precautions in such applications to ensure that the resource won't be purged while you're changing it.

To check for errors, call the `ResError` (page 1692) function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Resources.h`

Unique1ID

Gets a resource ID that's unique with respect to resources in the current resource file.

```
ResID Unique1ID (
   ResType theType
);
```

Parameters

theType

A resource type.

Return Value

A resource ID greater than 0 that isn't currently assigned to any resource of the specified type in the current resource file.

Discussion
You should use this function before adding a new resource to ensure that you don't duplicate a resource ID and override an existing resource.

To check for errors, call the `ResError` (page 1692) function.

For more information about restrictions on resource IDs for specific resource types, see `ResID` (page 1705).

In versions of system software earlier than System 7, this function may return a resource ID in the range 0 through 127, which is generally reserved for system resources. You should check that the resource ID returned is not in this range. If it is, call `UniqueID` again, and continue doing so until you get a resource ID greater than 127.

In System 7 and later versions, this function won't return a resource ID of less than 128.

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

Declared In
`Resources.h`

UniqueID

Gets a unique resource ID for a resource.

```
ResID UniqueID (
   ResType theType
);
```

Parameters

theType
A resource type.

Return Value
A resource ID greater than 0 that isn't currently assigned to any resource of the specified type in any open resource fork.

Discussion
You should use this function before adding a new resource to ensure that you don't duplicate a resource ID and override an existing resource.

To check for errors, call the `ResError` (page 1692) function.

For more information about restrictions on resource IDs for specific resource types, see `ResID` (page 1705).

In versions of system software earlier than System 7, this function may return a resource ID in the range 0 through 127, which is generally reserved for system resources. You should check that the resource ID returned is not in this range. If it is, call `UniqueID` again, and continue doing so until you get a resource ID greater than 127.

Version Notes
In System 7 and later versions, `UniqueID` won't return a resource ID of less than 128.

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

Declared In
Resources.h

UpdateResFile

Updates the resource map and resource data for a resource fork without closing it.

```
void UpdateResFile (
   ResFileRefNum refNum
);
```

Parameters

refNum

A file reference number for a resource fork. If there's no open resource fork with the given reference number, the function does nothing, and the ResError (page 1692) function returns the result code resNotFound. If the value of the refNum parameter is 0, it represents the System file's resource fork. If you call this function but the mapReadOnly attribute of the resource fork is set, the function does nothing, and the ResError function returns the result code resAttrErr.

Discussion
Given the reference number of a file whose resource fork is open, this function performs three tasks. The first task is to change, add, or remove resource data in the file's resource fork to match the resource map in memory. Changed resource data for each resource is written only if that resource's resChanged bit has been set by a successful call to the ChangedResource (page 1661) or AddResource (page 1660) function. The UpdateResFile function calls the WriteResource (page 1702) function to write changed or added resources to the resource fork.

The second task is to compact the resource fork, closing up any empty space created when a resource was removed, made smaller, or made larger. If a resource is made larger, the Resource Manager writes it at the end of the resource fork rather than at its original location. It then compacts the space occupied by the original resource data. The actual size of the resource fork is adjusted when a resource is removed or made larger, but not when a resource is made smaller.

The third task is to write the resource map in memory to the resource fork if your application has called the ChangedResource function for any resource listed in the resource map or if it has added or removed a resource. All changes to resource information in the resource map become permanent at this time; if you want any of these changes to be temporary, you must restore the original information before calling UpdateResFile.

Because the CloseResFile (page 1662) function calls UpdateResFile before it closes the resource fork, you need to call UpdateResFile directly only if you want to update the file without closing it.

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

Declared In
Resources.h

UseResFile

Sets the current resource file.

```
void UseResFile (
   ResFileRefNum refNum
);
```

Parameters

refNum

The file reference number for the resource fork which you wish to set as the current resource fork.

Return Value

The function searches the list of files whose resource forks have been opened for the file specified here. If the specified file is found, the Resource Manager sets the current resource file to the specified file. If there's no resource fork open for a file with that reference number, the function does nothing. To set the current resource file to the System file, use 0 here.

Discussion

Open resource forks are arranged as a linked list with the most recently opened resource fork at the beginning. When searching open resource forks, the Resource Manager starts with the most recently opened file. You can call this function to set the current resource file to a file opened earlier, and thereby start subsequent searches with the specified file. In this way, you can cause any files higher in the resource chain to be left out of subsequent searches.

When a new resource fork is opened, this action overrides previous calls to this function and the entire list is searched. For example, if five resource forks are opened in the order R0, R1, R2, R3, and R4, the search order is R4-R3-R2-R1-R0. Calling `UseResFile(R2)` changes the search order to R2-R1-R0; R4 and R3 are not searched. When the resource fork of a new file (R5) is opened, the search order becomes R5-R4-R3-R2-R1-R0.

You typically call the `CurResFile` (page 1664) function to get and save the current resource file, `UseResFile` to set the current resource file to the desired file, then (after you are finished using the resource) `UseResFile` to restore the current resource file to its previous value. Calling `UseResFile(0)` causes the Resource Manager to search only the System file's resource map. This is useful if you no longer wish to override a system resource with one by the same name in your application's resource fork.

Most of the Resource Manager functions assume that the current resource file is the file on whose resource fork they should operate or, in the case of a search, the file where they should begin. In general, the current resource file is the last one whose resource fork your application opened unless you specify otherwise.

The `FSpOpenResFile` (page 1671) and `HOpenResFile` (page 1686) functions, which also set the current resource file, override previous calls to `UseResFile`.

To check for errors, call the `ResError` (page 1692) function.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Simple DrawSprocket

Declared In

`Resources.h`

WritePartialResource

Writes part of a resource to disk when working with a small subsection of a large resource.

```
void WritePartialResource (
   Handle theResource,
   long offset,
   const void *buffer,
   long count
);
```

Parameters

theResource

A handle to the resource you wish to write to disk.

offset

The beginning of the resource subsection to write, measured in bytes from the beginning of the resource.

buffer

A pointer to the buffer containing the data to write. Your application is responsible for the buffer's memory management.

count

The length of the resource subsection to write.

Discussion

If the disk or the file is locked, the `ResError` (page 1692) function returns an appropriate File Manager result code. If you try to write past the end of a resource, the Resource Manager attempts to enlarge the resource. The `ResError` function returns the result code `writingPastEnd` if the attempt succeeds. If the Resource Manager cannot enlarge the resource, `ResError` returns an appropriate File Manager result code. If you pass an invalid value in the `offset` parameter, `ResError` returns the result code `inputOutOfBounds`.

This function tries to write the data from the buffer to disk. If the attempt is successful and the resource data (referenced through the resource's handle) is in memory, `ResError` returns the result code `resourceInMemory`. In this situation, be aware that the data of the resource subsection on disk matches the data from the buffer, not the resource data referenced through the resource's handle. If the attempt to write the data from the buffer to the disk fails, `ResError` returns an appropriate error.

When using partial resource functions, you should call the `SetResLoad` (page 1695) function, specifying `FALSE` for the `load` parameter, before you call the `GetResource` (page 1683) function. Doing so prevents the Resource Manager from reading the entire resource into memory. Be sure to restore the normal state by calling `SetResLoad` again, with the `load` parameter set to `TRUE`, immediately after you call `GetResource`.

If you read a partial resource into memory and then change its size, you must use the `SetResourceSize` (page 1696) function to change the entire resource's size on disk as necessary before you write the partial resource.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Resources.h`

WriteResource

Writes resource data in memory immediately to a file's resource fork.

```
void WriteResource (
   Handle theResource
);
```

Parameters

theResource

A handle to a resource. The function checks the `resChanged` attribute of this resource. If the `resChanged` attribute is set to 1, such as after a successful call to the `ChangedResource` (page 1661) or `AddResource` (page 1660) function, `WriteResource` writes the resource data in memory to the resource fork, then clears the `resChanged` attribute in the resource's resource map in memory.

If the resource is purgeable and has been purged, the function writes zero-length resource data to the resource fork. If the resource's `resProtected` attribute is set to 1, the function does nothing, and the `ResError` (page 1692) function returns the result code `noErr`. The same is true if the `resChanged` attribute is not set (that is, set to 0). If the given handle isn't a handle to a resource, `WriteResource` does nothing, and `ResError` returns the result code `resNotFound`.

Discussion

Note that this function does not write the resource's resource map entry to disk.

When your application calls `ChangedResource` or `AddResource`, the Resource Manager attempts to reserve disk space for the changed resource. If the modified resource data can't be written to the resource fork (for example, if there's not enough room on disk), the `resChanged` attribute is not set to 1. If this is the case and you call `WriteResource`, the Resource Manager won't know that the resource data has been changed. Thus, the function won't write the modified resource data to the resource fork and won't return an error. For this reason, always make sure that the `ResError` function returns the result code `noErr` after a call to `ChangedResource` or `AddResource`.

The resource fork is updated automatically when your application quits, when you call the `UpdateResFile` (page 1700) function, or when you call the `CloseResFile` (page 1662) function. Thus, you should call `WriteResource` only if you want to write just one or a few resources immediately.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Resources.h`

Callbacks

ResErrProcPtr

```
typedef void (*ResErrProcPtr) (
    OSErr thErr
);
```

If you name your function `MyResErrProc`, you would declare it like this:

```
void MyResErrProc (
    OSErr thErr
);
```

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

Declared In
`Resources.h`

ResourceEndianFilterPtr

```
typedef OSErr (*ResourceEndianFilterPtr) (
    Handle theResource,
    Boolean currentlyNativeEndian
);
```

If you name your function `MyResourceEndianFilter`, you would declare it like this:

```
OSErr MyResourceEndianFilter (
    Handle theResource,
    Boolean currentlyNativeEndian
);
```

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

Declared In
`Resources.h`

Data Types

ResAttributes

```
typedef short ResAttributes;
```

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

Declared In
`Resources.h`

ResErrUPP

```
typedef ResErrProcPtr ResErrUPP;
```

Discussion
For more information, see the description of the `ResErrProcPtr` (page 1703) callback function.

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

Declared In
Resources.h

ResFileAttributes

```
typedef short ResFileAttributes;
```

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

Declared In
Resources.h

ResFileRefNum

```
typedef short ResFileRefNum;
```

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

Declared In
Resources.h

ResID

Defines a unique identifier for a resource of a given type.

```
typedef short ResID;
```

Discussion
A resource is identified by its resource type and resource ID (or, optionally, its resource type and resource name). The IDs for resources used by the system software and those used by applications are assigned from separate ranges. By using these ranges correctly, you can avoid resource ID conflicts.

In general, system resources use IDs in the range –32767 through 127, and application resources must use IDs that fall between 128 and 32767. The IDs for some categories of resources, such as definition functions and font families, fall in different ranges or in ranges that are broken down for more specific purposes.

You can use a resource name instead of a resource ID to identify a resource of a given type. Like a resource ID, a resource name should be unique within each type. If you assign the same resource name to two resources of the same type, the second assignment of the name overrides the first, thereby making the first resource inaccessible by name. When comparing resource names, the Resource Manager ignores case (but does not ignore diacritical marks).

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

Declared In
Resources.h

ResType

Defines a unique identifier for a type of resource.

```
typedef FourCharCode ResType;
```

Discussion
The Resource Manager uses the resource type along with the resource ID to identify a resource. A resource type can be any sequence of four alphanumeric characters, including the space character.

You can define your own resource types, but they must not conflict with any of the standard resource types. When identifying resource types, the Resource Manager distinguishes between uppercase letters and their lowercase counterparts. Apple reserves for its own use all resource types that consist of all lowercase letters, all spaces, or all international characters (characters greater than $7F).

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

Declared In
`IOMacOSTypes.h`

Constants

Reference Number Constants

```
enum {
    kResFileNotOpened = -1,
    kSystemResFile = 0
};
```

Constants

`kResFileNotOpened`
Indicates the reference number returned as error when opening a resource file.
Available in Mac OS X v10.0 and later.
Declared in `Resources.h`.

`kSystemResFile`
Indicates the default reference number to the system file.
Available in Mac OS X v10.0 and later.
Declared in `Resources.h`.

Resource Attribute Bits

```
enum {
    resSysRefBit = 7,
    resSysHeapBit = 6,
    resPurgeableBit = 5,
    resLockedBit = 4,
    resProtectedBit = 3,
    resPreloadBit = 2,
    resChangedBit = 1,
};
```

Constants

`resSysRefBit`

If this attribute is set to 1, it is a system reference. If it is set to 0, it is a local reference.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

`resSysHeapBit`

This attribute indicates whether the resource is read into the system heap (`resSysHeapBit` attribute is set to 1) or your application's heap (`resSysHeapBit` attribute is set to 0).

If you are setting your resource's attributes with `SetResAttrs`, you should set this bit to 0 for your application's resources. Note that if you do set the `resSysHeapBit` attribute to 1 and the resource is too large for the system heap, the bit is cleared and the resource is read into the application heap.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

`resPurgeableBit`

If this attribute is set to 1, the resource is purgeable if it's 0, the resource is nonpurgeable. However, do not use `SetResAttrs` to make a purgeable resource nonpurgeable.

Because a locked resource is nonrelocatable and nonpurgeable, the `resLockedBit` attribute overrides the `resPurgeableBit` attribute.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

`resLockedBit`

If this attribute is 1, the resource is nonpurgeable regardless of whether `resPurgeableBit` is set. If it's 0, the resource is purgeable or nonpurgeable depending on the value of the `resPurgeableBit` attribute.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

`resProtectedBit`

If this attribute is set to 1, your application can't use Resource Manager functions to change the resource ID or resource name, modify the resource contents, or remove the resource from its resource fork. However, you can use the `SetResAttrs` function to remove this protection. Note that this attribute change takes effect immediately.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

`resPreloadBit`
: If this attribute is set to 1, the Resource Manager reads the resource's resource data into memory immediately after opening its resource fork. You can use this setting to make multiple resources available for your application as soon as possible, rather than reading each one into memory individually. If both the `resPreloadBit` attribute and the `resLockedBit` attribute are set, the Resource Manager loads the resource as low in the heap as possible.

  Available in Mac OS X v10.0 and later.

  Declared in `Resources.h`.

`resChangedBit`
: If this attribute is set to 1, the resource has been changed. If it's 0, the resource hasn't been changed. This attribute is used only while the resource map is in memory. The `resChangedBit` attribute must be 0 in the resource fork on disk.

  Do not use `SetResAttrs` to set the `resChangedBit` attribute. Be sure the `attrs` parameter passed to `SetResAttrs` doesn't change the current setting of this attribute. To set the `resChangedBit` attribute, call the `ChangedResource` function.

  Available in Mac OS X v10.0 and later.

  Declared in `Resources.h`.

Discussion
The `SetResAttrs` (page 1693) and `GetResAttrs` (page 1681) functions use these constants to refer to each attribute.

Resource Attribute Masks

```
enum {
    resSysHeap = 64,
    resPurgeable = 32,
    resLocked = 16,
    resProtected = 8,
    resPreload = 4,
    resChanged = 2,
};
```

Constants
`resSysHeap`
: Use to set or test for the `resSysHeapBit`.

  Available in Mac OS X v10.0 and later.

  Declared in `Resources.h`.

`resPurgeable`
: Use to set or test for the `resPurgeableBit`.

  Available in Mac OS X v10.0 and later.

  Declared in `Resources.h`.

`resLocked`
: Use to set or test for the `resLockedBit`.

  Available in Mac OS X v10.0 and later.

  Declared in `Resources.h`.

`resProtected`
Use to set or test for the `resProtectedBit`.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

`resPreload`
Use to set or test for the `resPreloadBit`.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

`resChanged`
Use to set or test for the `resChangedBit`.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

Resource Chain Location

Specify the location of the resource chain.

```
typedef SInt16 RsrcChainLocation
enum {
    kRsrcChainBelowSystemMap = 0,
    kRsrcChainBelowApplicationMap = 1,
    kRsrcChainAboveApplicationMap = 2,
    kRsrcChainAboveAllMaps = 4
};
```

Constants

`kRsrcChainBelowSystemMap`
Indicates the resource chain is below the system's resource map.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

`kRsrcChainBelowApplicationMap`
Indicates the resource chain is below the application's resource map.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

`kRsrcChainAboveApplicationMap`
Indicates the resource chain is above the application's resource map.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

`kRsrcChainAboveAllMaps`
Indicates the resource chain is above all resource maps.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

Discussion

These constants and data type are for use with the Resource Manager chain manipulation routines under Carbon.

Resource Fork Attribute Bits

```
enum {
    mapReadOnlyBit = 7,
    mapCompactBit = 6,
    mapChangedBit = 5
};
```

Constants

`mapReadOnlyBit`

If this bit is set to 1, the Resource Manager doesn't write anything to the resource fork on disk. It also doesn't check whether the resource data can be written to disk when the resource map is modified. When this attribute is set to 1, the `ChangedResource` (page 1661) and `WriteResource` (page 1702) functions do nothing, but the function `ResError` (page 1692) returns the result code `noErr`.

If you set the `mapReadOnlyBit` attribute but later clear it, the resource data is written to disk even if there's no room for it. This operation may destroy the resource fork.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

`mapCompactBit`

If this bit is set to 1, the Resource Manager compacts the resource fork when it updates the file. The Resource Manager sets this attribute when a resource is removed or when a resource is made larger and thus must be written at the end of a resource fork. You may want to set the `mapCompactBit` attribute to force the Resource Manager to compact a resource fork when your changes have made resources smaller.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

`mapChangedBit`

If this bit is set to 1, the Resource Manager writes the resource map to disk when the file is updated. For example, you can set `mapChangedBit` if you've changed resource attributes only and don't want to call the `ChangedResource` (page 1661) function because you don't want to write the resource data to disk.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

Resource Fork Attribute Masks

```
enum{
    mapReadOnly = 128,
    mapCompact = 64,
    mapChanged = 32
};
```

Constants

`mapReadOnly`

Use to set or test for the `mapReadOnlyBit`.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

`mapCompact`

Use to set or test for the `mapCompactBit`.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

`mapChanged`

Use to set or test for the `mapChangedBit`.

Available in Mac OS X v10.0 and later.

Declared in `Resources.h`.

Result Codes

The most common result codes returned by Resource Manager are listed in the table below. The Resource Manager may also return the following result codes: `noErr` (0), `dirFulErr` (-33), `dskFulErr` (-34), `nsvErr` (-35), `ioErr` (-36), `bdNamErr` (-37), `eofErr` (-39), `tmfoErr` (-42), `fnfErr` (-43), `wPrErr` (-44), `fLckdErr` (-45), `vLckdErr` (-46), `dupFNErr` (-48), `opWrErr` (-49), `permErr` (-54), `extFSErr` (-58), `memFullErr` (-108), `dirNFErr` (-120).

Result Code	Value	Description
`badExtResource`	-185	The extended resource has a bad format. Available in Mac OS X v10.0 and later.
`CantDecompress`	-186	Can't decompress a compressed resource. Available in Mac OS X v10.0 and later.
`resourceInMemory`	-188	The resource is already in memory. Available in Mac OS X v10.0 and later.
`writingPastEnd`	-189	Writing past the end of file. Available in Mac OS X v10.0 and later.
`inputOutOfBounds`	-190	The offset or count is out of bounds. Available in Mac OS X v10.0 and later.
`resNotFound`	-192	The resource was not found. Available in Mac OS X v10.0 and later.
`resFNotFound`	-193	The resource file was not found. Available in Mac OS X v10.0 and later.
`addResFailed`	-194	The `AddResource` function failed. Available in Mac OS X v10.0 and later.
`rmvResFailed`	-196	The `RemoveResource` function failed. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`resAttrErr`	-198	The attribute is inconsistent with the operation. Available in Mac OS X v10.0 and later.
`mapReadErr`	-199	The map is inconsistent with the operation. Available in Mac OS X v10.0 and later.

Script Manager Reference (Not Recommended)

Framework: CoreServices/CoreServices.h

Declared in Script.h

Overview

Important: The Script Manager is deprecated as of Mac OS X v10.5. Instead, you should update your application to handle Unicode text using the facilities of the Cocoa system (see *Text System Overview*) or Core Text (see *Core Text Programming Guide*). See also *Internationalization Programming Topics*.

The Script Manager makes script systems available and coordinates the interaction between many parts of the Mac OS and those available script systems. A script system (or script for short) is a collection of resources that provides for the representation of a particular writing system.

The Script Manager also provides several services directly to your application. Through them you can get information about the current text environment, modify that environment, and perform a variety of text-handling tasks.

The Script Manager has evolved through several versions. It started with sole responsibility for all international-compatibility and multilingual text issues, but as more power and features have been added, many of its specific functions have been moved to the other parts of system software.

For many text-related tasks, the Script Manager's role is transparent when you make a script-aware Text Utilities or QuickDraw call while processing text, that routine may get the information it needs through the Script Manager. For example, when you call the QuickDraw function `DrawText` to draw a line of text, `DrawText` in turn calls the Script Manager to determine which script system your text belongs to before drawing it. In other situations you may need to call the Script Manager explicitly, to properly interpret the text you are processing.

Carbon supports most Script Manager functions. However, Apple recommends that whenever possible you should replace Script Manager calls with the appropriate Unicode functionality. For more information, see Unicode Utilities Reference and Supporting Unicode Input.

See also the `KeyScript` function documentation.

Functions by Task

Analyzing Characters

`CharacterByteType` (page 1716) Deprecated in Mac OS X v10.4
Identifies a byte in a text buffer as a single-byte character or as the first or second byte of a double-byte character. (Deprecated. You should update your application to handle Unicode text. There is no replacement function because Unicode handles encoding in a different manner.)

`CharacterType` (page 1717) Deprecated in Mac OS X v10.4
Returns a variety of information about the character represented by a given byte, including its type, class, orientation, direction, case, and size (in bytes). (Deprecated. You should update your application to handle Unicode text. There is no replacement function because Unicode handles encoding in a different manner.)

`FillParseTable` (page 1720) Deprecated in Mac OS X v10.4
Helps your application to quickly process a buffer of mixed single-byte and double-byte characters. (Deprecated. You should update your application to handle Unicode text. There is no replacement function because Unicode handles encoding in a different manner.)

Checking and Setting Script Manager Variables

`GetScriptManagerVariable` (page 1724) Deprecated in Mac OS X v10.5
Retrieves the value of the specified Script Manager variable. (Deprecated. The replacement for this function depends on the selector used with it, as described in the Special Considerations section.)

`SetScriptManagerVariable` (page 1731) Deprecated in Mac OS X v10.5
Sets the specified Script Manager variable to the value of the input parameter. (Deprecated. This is mainly used to set the value of variables that control the internal operation of the Script Manager (selectors `smIntlForce` and `smGenFlags`), and therefore there is no modern replacement.)

Checking and Setting Script Variables

`GetScriptVariable` (page 1726) Deprecated in Mac OS X v10.5
Retrieves the value of the specified script variable from the specified script system. (Deprecated. The replacement for this function depends on the selector used with it, as described in the Special Considerations section.)

`SetScriptVariable` (page 1732) Deprecated in Mac OS X v10.5
Sets the specified script variable for the specified script system to the value of the input parameter. (Deprecated. The replacement for this function depends on the purpose for which it is used, as described in the Special Considerations section.)

Checking and Setting the System Direction

`GetSysDirection` (page 1727) Deprecated in Mac OS X v10.4
Returns the current value of `SysDirection`, the global variable that determines the system direction (primary line direction). (Deprecated. This function does not return anything useful in Mac OS X.)

`SetSysDirection` (page 1733) Deprecated in Mac OS X v10.4
Sets the value of `SysDirection`, the global variable that determines the system direction (primary line direction). (Deprecated. There is no replacement because this function is no longer needed in Mac OS X.)

Determining Script Codes From Font Information

`FontScript` (page 1721) Deprecated in Mac OS X v10.4
Returns the script code for the current script (usually the font script). (Deprecated. Use `ATSFontFamilyGetEncoding` instead.)

`FontToScript` (page 1722) Deprecated in Mac OS X v10.4
Translates a font family ID number into its corresponding script code, if that script system is currently enabled. (Deprecated. Use `ATSFontFamilyGetEncoding` instead.)

`IntlScript` (page 1728) Deprecated in Mac OS X v10.4
Identifies the script system used by the Text Utilities date-formatting, time-formatting, and string-sorting functions. (Deprecated. Use `ATSFontFamilyGetEncoding` instead.)

Directly Accessing International Resources

`GetIntlResource` (page 1722) Deprecated in Mac OS X v10.5
Returns a handle to one of the international resources. (Deprecated. The replacement for this function depends on the purpose for which it is used, as described in the Special Considerations section.)

`ClearIntlResourceCache` (page 1719) Deprecated in Mac OS X v10.4
Clears the application's international resources cache, which contains the resource ID numbers of the string-manipulation (`'itl2'`) and tokens (`'itl4'`) resources for the current script. (Deprecated. There is no replacement because this function is no longer needed in Mac OS X.)

`GetIntlResourceTable` (page 1723) Deprecated in Mac OS X v10.4
Obtains a specific word-selection, line-break, number-parts, untoken, or whitespace table from the appropriate international resource. (Deprecated. There is no replacement because this function is no longer needed in Mac OS X.)

Converting Text

`IntlTokenize` (page 1729) Deprecated in Mac OS X v10.4
Allows your application to convert text into a sequence of language-independent tokens. (Deprecated. There is no replacement because this function is no longer needed in Mac OS X.)

`TransliterateText` (page 1733) Deprecated in Mac OS X v10.4
Converts characters from one subscript to the closest possible approximation in a different subscript within the same double-byte script system. (Deprecated. Use `CFStringUppercase` instead.)

Functions

CharacterByteType

Identifies a byte in a text buffer as a single-byte character or as the first or second byte of a double-byte character. (Deprecated in Mac OS X v10.4. You should update your application to handle Unicode text. There is no replacement function because Unicode handles encoding in a different manner.)

```
short CharacterByteType (
   Ptr textBuf,
   short textOffset,
   ScriptCode script
);
```

Parameters

textBuf

A pointer to a text buffer containing the byte to be identified.

textOffset

The offset to the byte to be identified. The offset is measured in bytes; the first byte has an offset of 0.

script

A value that specifies the script system of the text in the buffer. Constants for all defined script codes are listed on "Meta Script Codes" (page 1753). To specify the font script, pass `smCurrentScript` in this parameter.

Return Value

One of three identifications: a single-byte character, the first byte of a double-byte character, or the second byte of a double-byte character. The first byte of a double-byte character—the one at the lower offset in memory—is the high-order byte the second byte of a double-byte character—the one at the higher offset—is the low-order byte. This is the same order in which text is processed and numbers are represented.

Discussion

The script system associated with the character you wish to examine must be enabled in order for the function to provide useful information. For example, if only the Roman script system is available and you attempt to identify a byte in a run of double-byte characters, the `CharacterByteType` function returns 0, indicating that the byte is a single-byte character.

For single-byte script systems, the character-type tables reside in the string-manipulation (`'itl2'`) resource and reflect region-specific or language-specific differences in uppercase conventions.

For double-byte script systems, the character-type tables reside in the encoding/rendering (`'itl5'`) resource, not the string-manipulation resource. Whenever you call `CharacterByteType`, the necessary character-set encoding information is taken from the encoding/rendering resource. You cannot use the `GetIntlResource` function to access double-byte character-type tables directly.

From byte value alone, it is not possible to distinguish the second byte of a double-byte character from a single-byte character. `CharacterByteType` differentiates the second byte of a double-byte character from a single-byte character by assuming that the byte at offset 0 is the first byte of a character. With that assumption, it then sequentially identifies the size and starting position of each character in the buffer up to `textOffset`.

Special Considerations

If you specify `smCurrentScript` for the `script` parameter, the value returned by `CharacterByteType` can be affected by the state of the font force flag. It is unaffected by the state of the international resources selection flag.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Script.h`

CharacterType

Returns a variety of information about the character represented by a given byte, including its type, class, orientation, direction, case, and size (in bytes). (Deprecated in Mac OS X v10.4. You should update your application to handle Unicode text. There is no replacement function because Unicode handles encoding in a different manner.)

```
short CharacterType (
   Ptr textBuf,
   short textOffset,
   ScriptCode script
);
```

Parameters

textBuf

A pointer to a text buffer containing the character to be examined.

textOffset

The offset to the location of the character to be examined. (It can be an offset to either the first or the second byte of a double-byte character.) Offset is in bytes; the first byte of the first character has an offset of 0.

script

A value that specifies the script system the byte belongs to. Constants for all defined script codes are listed in "Meta Script Codes" (page 1753). To specify the font script, pass `smCurrentScript` in this parameter.

Return Value

An integer bit field that provides information about the requested character.

Discussion

The `CharacterType` return value is an integer bit field that provides information about the requested character. The field has the following format:

- Bit range 0–3 (Type). The character types.
- Bit range 4–7. Reserved.
- Bit Range 8–11 (Class). Character classes (i.e., subtypes).
- Bit 12 (Orientation). Horizontal or vertical.
- Bit 13 (Direction). Left or right. In double-byte script systems, bit 13 indicates whether or not the character is part of the main character set (not a user-defined character).

- Bit 14 (Case). Uppercase or lowercase.
- Bit 15 (Size). single-byte or double-byte.

The script system associated with the character you wish to examine must be enabled in order for any of these three functions to provide useful information.

For single-byte script systems, the character-type tables reside in the string-manipulation (`'itl2'`) resource and reflect region-specific or language-specific differences in uppercase conventions. The `CharacterType` function gets the tables from the string-manipulation resource using the `GetIntlResource` function.

For double-byte script systems, the character-type tables reside in the encoding/rendering (`'itl5'`) resource, not the string-manipulation resource. Whenever you call `CharacterType`, the necessary character-set encoding information is taken from the encoding/rendering resource. You cannot use the `GetIntlResource` function to access double-byte character-type tables directly.

The Script Manager defines the recognized character types, character classes, and character modifiers (bits 12–15), with constants to describe them. The `CharacterType` field masks are described in "Character Type Field Masks" (page 1746).

The Script Manager also defines a set of masks with which you can isolate each of the fields in the `CharacterType` return value. If you perform an `AND` operation with the `CharacterType` result and the mask for a particular field, you select only the bits in that field. Once you've done that, you can test the result, using the constants that represent the possible results.

The function `CharacterType` calls `CharacterByteType` to determine whether the byte at `textOffset` is a single-byte character or the first byte or second byte of a double-byte character. The larger the text buffer, the longer `CharacterByteType` takes to execute. To be most efficient, place the pointer `textBuf` at the beginning of the character of interest before calling `CharacterType`. (If you want to be compatible with older versions of `CharacterType`, also set `textOffset` to 1, rather than 0, for double-byte characters.)

Special Considerations

The function `CharacterType` may move memory; your application should not call this function at interrupt time.

If you specify `smCurrentScript` for the `script` parameter, `CharacterType` always assumes that the text in the buffer belongs to the font script. It is unaffected by the state of the font force flag or the international resources selection flag.

For single-byte script systems, the character-type tables are in the string-manipulation (`'itl2'`) resource. For double-byte script systems, they are in the encoding/rendering (`'itl5'`) resource. If the appropriate resource does not include these tables, `CharacterType` exits without doing anything.

Some Roman fonts (for example, Symbol) substitute other characters for the standard characters in the Standard Roman character set. Since the Roman script system `CharacterType` function assumes the Standard Roman character set, it may return inappropriate results for nonstandard characters.

Version Notes

In versions of system software earlier than 7.0, the `textOffset` parameter to the `CharacterType` function must point to the second byte of a double-byte character.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Script.h`

ClearIntlResourceCache

Clears the application's international resources cache, which contains the resource ID numbers of the string-manipulation (`'itl2'`) and tokens (`'itl4'`) resources for the current script. (Deprecated in Mac OS X v10.4. There is no replacement because this function is no longer needed in Mac OS X.)

Not recommended

```
void ClearIntlResourceCache (
   void
);
```

Discussion
At application launch, the script management system sets up an international resources cache for the application. The cache contains the resource ID numbers of the string-manipulation and tokens resources for all enabled scripts.

If you provide your own string manipulation or tokens resource to replace the default for a particular script, call `ClearIntlResourceCache` at launch to ensure that your supplied resource is used instead of the script system's `'itl2'` or `'itl4'` resource.

The current default ID numbers for a script system's `'itl2'` and `'itl4'` resources are stored in its script variables. You can read and modify these values with the `GetScriptVariable` and `SetScriptVariable` functions using the selectors `smScriptSort` (for the `'itl2'` resource) and `smScriptToken` (for the `'itl4'` resource). Before calling `ClearIntlResourceCache`, you should set the script's default ID number to the ID of the resource that you are supplying.

If the international resources selection flag is `TRUE`, the ID numbers of your supplied resources must be in the system script range. Otherwise, the IDs must be in the range of the current script.

If you use the `SetScriptVariable` function to change the value of the `'itl2'` or `'itl4'` resource ID and then call `ClearIntlResourceCache` to flush the cache, be sure to restore the original resource ID before your application quits.

Special Considerations

The function `ClearIntlResourceCache` 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
`Script.h`

FillParseTable

Helps your application to quickly process a buffer of mixed single-byte and double-byte characters. (Deprecated in Mac OS X v10.4. You should update your application to handle Unicode text. There is no replacement function because Unicode handles encoding in a different manner.)

```
Boolean FillParseTable (
   CharByteTable table,
   ScriptCode script
);
```

Parameters

table

A 256-byte table to be filled in by `FillParseTable`.

script

A value that specifies the script system the parse table belongs to. Constants for all defined script codes are listed in "Meta Script Codes" (page 1753). To specify the font script, pass `smCurrentScript` in this parameter.

Return Value

If you specify `smCurrentScript` for the `script` parameter, the value returned by `FillParseTable` can be affected by the state of the font force flag. It is unaffected by the international resources selection flag.

Discussion

Before calling `FillParseTable`, allocate space for a 256-byte table to pass to the function in the table parameter.

This function returns a 256-byte table that distinguishes the character codes of all possible single-byte characters from the first (high-order) byte values of all possible double-byte characters in the specified script system. The script system associated with the character you wish to examine must be enabled in order for any of these three functions to provide useful information.

For single-byte script systems, the character-type tables reside in the string-manipulation (`'itl2'`) resource and reflect region-specific or language-specific differences in uppercase conventions.

For double-byte script systems, the character-type tables reside in the encoding/rendering (`'itl5'`) resource, not the string-manipulation resource. Whenever you call `FillParseTable`, the necessary character-set encoding information is taken from the encoding/rendering resource. You cannot use the `GetIntlResource` function to access double-byte character-type tables directly.In every script system, double-byte characters have distinctive high-order (first) bytes that allow them to be distinguished from single-byte characters. `FillParseTable` fills a 256-byte table, conceptually equivalent to a single-byte character-set table, with values that indicate, byte-for-byte, whether the character-code value represented by that byte index is the first byte of a double-byte character. An entry in the `CharByteTable` is 0 for a single-byte character and 1 for the first byte of a double-byte character.

If your application is processing mixed characters, it can use the table to identify the locations of the double-byte characters as it makes a single pass through the text, rather than having to call `CharacterByteType` or `CharacterType` for each byte of the text buffer in turn. `CharacterByteType` and `CharacterType` start anew at the beginning of the text buffer each time they are called, tracking character positions up to the offset of the byte to be analyzed.

Special Considerations

`FillParseTable` may move memory; your application should not call this function at interrupt time.

The table defined by `CharByteTable` is not dynamic; it does not get updated when the current font changes. You need to call it separately for each script run in your 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
`Script.h`

FontScript

Returns the script code for the current script (usually the font script). (Deprecated in Mac OS X v10.4. Use `ATSFontFamilyGetEncoding` instead.)

```
short FontScript (
   void
);
```

Parameters

Return Value
A script code. All recognized script codes and their defined constants are listed in "Meta Script Codes" (page 1753). `FontScript` returns only explicit script codes (□). If the font of the active graphics port is Roman and the font force flag is `TRUE`, the script code returned is that of the system script and the script-forced result flag is set to `TRUE`. If the font of the active graphics port is non-Roman, the state of the font force flag is ignored. If the script system corresponding to the font of the active graphics port is not installed and enabled, the script code returned is that of the system script and the script-defaulted result flag is set to `TRUE`.

Discussion
The information about the script code is subject to two control flags—the font force flag and the international resources selection flag. You can test and set these flags with the `GetScriptManagerVariable` and `SetScriptManagerVariable` selectors `smFontForce` and `smIntlForce`.

The function starts by initializing two result flags, the script-forced result flag and the script-defaulted result flag, to `FALSE`. These flags are Script Manager variables, accessed through the `GetScriptManagerVariable` function selectors `smForced` and `smDefault`.

Special Considerations

`FontScript` 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
`Script.h`

FontToScript

Translates a font family ID number into its corresponding script code, if that script system is currently enabled. (Deprecated in Mac OS X v10.4. Use `ATSFontFamilyGetEncoding` instead.)

```
short FontToScript (
   short fontNumber
);
```

Parameters

fontNumber

A font family ID number.

Return Value

A script code. All recognized script codes and their defined constants are listed in "Meta Script Codes" (page 1753). `FontToScript` returns only explicit script codes (□). If `fontNumber` is in the Roman range and the font force flag is `TRUE`, the script code returned is that of the system script and the script-forced result flag is set to `TRUE`. If `fontNumber` is in the non-Roman range, the state of the font force flag is ignored. If the script system corresponding to `fontNumber` is not enabled, the script code returned is that of the system script and the script-defaulted result flag is set to `TRUE`.

Discussion

The information about the script code is subject to two control flags—the font force flag and the international resources selection flag. You can test and set these flags with the `GetScriptManagerVariable` and `SetScriptManagerVariable` selectors `smFontForce` and `smIntlForce`.

The function starts by initializing two result flags, the script-forced result flag and the script-defaulted result flag, to `FALSE`. These flags are Script Manager variables, accessed through the `GetScriptManagerVariable` function selectors `smForced` and `smDefault`.

Do not use the function `FontToScript` to convert resource IDs to scripts codes.

Special Considerations

`FontToScript` 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

`Script.h`

GetIntlResource

Returns a handle to one of the international resources. (Deprecated in Mac OS X v10.5. The replacement for this function depends on the purpose for which it is used, as described in the Special Considerations section.)

```
Handle GetIntlResource (
   short theID
);
```

Parameters

theID

Contains an integer (0, 1, 2, 4, or 5 respectively for the `'itl0'`, `'itl1'`, `'itl2'`, `'itl4'`, and `'itl5'` resources) to identify the type of the desired international resource.

Return Value

A handle to the correct resource of the requested type. The resource returned is that of the current script, which is either the font script or the system script. The resource is of one of the following types: numeric-format (`'itl0'`), long-date-format (`'itl1'`), string-manipulation (`'itl2'`), tokens (`'itl4'`), or encoding/rendering (`'itl5'`). If `GetIntlResource` cannot return the requested resource, it returns a `NULL` handle and sets the global variable `resErr` to the appropriate error code.

Special Considerations

Depending on the information that this function was called to obtain, it can be replaced by the use of `CFLocaleCopyCurrent` to get an appropriate `CFLocaleRef` followed by one of the following:

- `CFLocaleGetValue` with keys such as `kCFLocaleUsesMetricSystem`, `kCFLocaleDecimalSeparator`, `kCFLocaleCurrencySymbol`.
- `CFDateFormatterCreate` to get an appropriate `CFDateFormatterRef` object, followed by `CFDateFormatterCopyProperty` with keys such as `kCFDateFormatterMonthSymbols`, `kCFDateFormatterWeekdaySymbols`, and `kCFDateFormatterAMSymbol`.
- `CFNumberFormatterCreate` to get an appropriate `CFNumberFormatterRef` object, followed by `CFNumberFormatterCopyProperty` with keys such as `kCFNumberFormatterCurrencyDecimalSeparator`, `kCFNumberFormatterMinusSign`, `kCFNumberFormatterPercentSymbol`, and `kCFNumberFormatterNegativePrefix`.

`GetIntlResource` may move memory; your application should not call this function at interrupt time.

Carbon Porting Notes

While the return type of this function remains a Handle, in Mac OS X it returns an ordinary memory handle instead of a resource handle.

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

`Script.h`

GetIntlResourceTable

Obtains a specific word-selection, line-break, number-parts, untoken, or whitespace table from the appropriate international resource. (Deprecated in Mac OS X v10.4. There is no replacement because this function is no longer needed in Mac OS X.)

```
void GetIntlResourceTable (
   ScriptCode script,
   short tableCode,
   Handle *itlHandle,
   long *offset,
   long *length
);
```

Parameters

script

A script code, the value that specifies a particular script system. Constants for all defined script codes are listed in "Meta Script Codes" (page 1753).

tableCode

A number that specifies which table is requested. The constants for `tableCode` are detailed in "Table Selectors" (page 1778).

itlHandle

On return, a handle to the string-manipulation (`'itl2'`) or tokens (`'itl4'`) resource containing the table specified in the `tableCode` parameter. If the script system whose table is requested is not available, `GetIntlResourceTable` returns a `NULL` handle.

offset

On return, a pointer to the offset (in bytes) to the specified table from the beginning of the resource.

length

On return, a pointer to the size of the table (in bytes).

Discussion

When you provide a script code in the `script` parameter, and a table code in the `tableCode` parameter, `GetIntlResourceTable` returns a handle to the string-manipulation resource or tokens resource containing that table, the offset of the specified table from the beginning of the resource, and the length of the table.

If you wish to manipulate the contents of the table you have requested, use the size returned in the `length` parameter to allocate a buffer, and perform a block move of the table's contents into that buffer.

Special Considerations

`GetIntlResourceTable` 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

`Script.h`

GetScriptManagerVariable

Retrieves the value of the specified Script Manager variable. (Deprecated in Mac OS X v10.5. The replacement for this function depends on the selector used with it, as described in the Special Considerations section.)

```
long GetScriptManagerVariable (
   short selector
);
```

Parameters

selector

A value that specifies a particular Script Manager variable. To specify the Script Manager variable whose value you need, use one of the selector constants listed in "Script Manager Selectors" (page 1768).

Return Value

The current value of the specified Script Manager variable or 0 if the selector is invalid. For some valid selectors, 0 may also be a valid return value. For example, when you call `GetScriptManagerVariable` with a selector value of `smRegionCode` on a version of Macintosh system software that has been localized for the United States, it returns 0. Although `GetScriptManagerVariable` always returns a long integer, the actual value may be a long integer, standard integer, or signed byte. If the value is not a long integer, it is stored in the low-order word or byte of the long integer returned by `GetScriptManagerVariable`; the remaining bytes are set to 0.

Discussion

The Script Manager maintains a set of variables that control general settings of the text environment, including the identity of the system script and the keyboard script, and the settings of the font force flag and the international resources selection flag.

You may want access to the Script Manager variables in order to understand the current environment or to modify it.

Special Considerations

The replacement for this function depends on the selector used with it. Many of the selectors refer to information that is not meaningful on a Unicode system or refer to details of the Script Manager itself; in general there are no replacements for these. Selectors that have meaningful replacements are shown in the following list. These are not direct replacements; they provide analogous but more modern functionality.

`smSysScript`. To obtain a text encoding for the legacy Mac OS encoding associated with the user's preferred user interface language or with the application's default text encoding, use `CFStringGetSystemEncoding` or `GetApplicationTextEncoding`. Sometimes `smSysScript` is just used to get a script code to pass to `GetScriptVariable` (page 1726); in this case the replacements for `GetScriptVariable` (page 1726) selectors may provide more information.

`smKeyScript`. To obtain the intended language associated with the user's current keyboard input source (plus other languages that can be input using it), use `TISCopyCurrentKeyboardInputSource` to get that input source, then pass it to `TISGetInputSourceProperty` with the `kTISPropertyInputSourceLanguages` key.

`smKCHRCache`. To obtain the key layout data for the keyboard layout currently in use, use `TISCopyCurrentKeyboardLayoutInputSource` to get that input source, then pass it to `TISGetInputSourceProperty` with the `kTISPropertyUnicodeKeyLayoutData` key (this returns `'uchr'` Unicode ayout data only; it will not return any data for keyboard layouts that only have `'KCHR'` data).

`smRegionCode`. To obtain the locale associated with the user's preferred formats (for dates, times, numbers, and so on) use the following code:

```
CFStringRef curLocaleStringRef = NULL;
localeRef = CFLocaleCopyCurrent();
if (localeRef) {
    curLocaleStringRef = CFLocaleGetIdentifier(localeRef);
    CFRelease(localeRef);
```

```
}
```

To obtain the user's preferred user interface language, use the following line of code:

```
CFArrayRef langArray = (CFArrayRef)CFPreferencesCopyAppValue(CFSTR("AppleLanguages"),
kCFPreferencesCurrentApplication);
```

The first entry in `langArray` indicates the preferred language. See also `CFLocaleCopyPreferredLanguages`.

Selectors that have no meaningful replacement on a Unicode system include `smEnabled`, `smBidirect`, and `smDoubleByte`. Selectors that pertain to internal operation of the Script Manager itself and thus have no meaningful replacement include `smVersion`, `smMunged`, `smPrint`, and `smSysRef`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Script.h`

GetScriptVariable

Retrieves the value of the specified script variable from the specified script system. (Deprecated in Mac OS X v10.5. The replacement for this function depends on the selector used with it, as described in the Special Considerations section.)

```
long GetScriptVariable (
   short script,
   short selector
);
```

Parameters

script

A value that specifies the script system whose variable you are accessing. Use one of the script-code constants listed in "Meta Script Codes" (page 1753).

selector

A value that specifies a particular script variable. Use one of the selector constants listed in "Script Variable Selectors" (page 1773). Valid selector values are defined by each script system.

Return Value
0 if the selector value is invalid or if the specified script system is not installed. For some valid selectors, 0 may also be a valid return value. For example, calling `GetScriptVariable` with a selector of `smScriptLang` on a version of Macintosh system software that has been localized for the United States returns 0. Although `GetScriptVariable` always returns a long integer, the actual value may be a long integer, standard integer, or signed byte. If the value is not a long integer, it is stored in the low-order word or byte of the long integer returned by `GetScriptVariable`; the remaining bytes are set to 0.

Discussion
Each enabled script system maintains a set of variables that control the current settings of that script system, including the ID numbers of its international resources, its preferred fonts and font sizes, and its primary line direction.

Special Considerations

The replacement for this function depends on the selector used with it. Many of the selectors refer to information that is not meaningful on a Unicode system or refer to details of the Script Manager itself; in general there are no replacements for these. Selectors that have meaningful replacements are shown in the following list. These are not direct replacements; they provide analogous but more modern functionality.

`smScriptLang`. This was typically used with the system script to determine the system language. Instead, to obtain the user's preferred user interface language, use the following line of code:

```
CFArrayRef langArray = (CFArrayRef)CFPreferencesCopyAppValue(CFSTR("AppleLanguages"),
kCFPreferencesCurrentApplication);
```

The first entry in `langArray` indicates the preferred language. See also `CFLocaleCopyPreferredLanguages`.

Font selectors `smScriptSysFond`, `smScriptSysFondSize`, `smScriptAppFond`, `smScriptAppFondSize`, `smScriptMonoFondSize`, `smScriptPrefFondSize`, `smScriptSmallFondSize`, and `smScriptHelpFondSize`. On Mac OS X you generally do not need to worry about setting an appropriate font based on character script to ensure that characters are displayed correctly; Unicode encoding and font fallbacks (to automatically find a font that can display a character) take care of this. However, for cases where you do need to do this (such as Carbon applications that handle non-Unicode text), the Core Text function `CTFontCreateUIFontForLanguage` (available in Mac OS X v10.5 and later) provides a way to get a `CTFontRef` object for a specified language and user interface use.

Script resource ID selectors `smScriptNumber`, `smScriptDate`, `smScriptSort`, and `smScriptToken`. These were used in several ways. Sometimes they were used to get a resource ID so specific fields in the resource could be examined (for example, to determine the appropriate decimal separator or time format). For this use `CFLocaleGetValue` can now be used with an appropriate key (for example, `kCFLocaleDecimalSeparator`) to get similar information (much of the information associated with the resource specified by `smScriptToken` is not relevant for a Unicode system). Another use was to get a resource ID (or a handle) to pass to some other system function. For text sorting, this is replaced by the collation functionality in CFString. For formatting of times, dates, and numbers, this is replaced by functionality in CFLocale, CFDateFormatter, CFNumberFormatter.

`smScriptKeys`. To determine an appropriate keyboard input source for a particular language, use `TISCopyInputSourceForLanguage`.

`smScriptIcon`. To obtain an icon for a particular keyboard input source, use `TISGetInputSourceProperty` with the `kTISPropertyIconRef` or `kTISPropertyIconImageURL` key.

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

`Script.h`

GetSysDirection

Returns the current value of `SysDirection`, the global variable that determines the system direction (primary line direction). (Deprecated in Mac OS X v10.4. This function does not return anything useful in Mac OS X.)

```
short GetSysDirection (
   void
);
```

Parameters

Return Value
The current value of `SysDirection`: 0 if the system direction is left-to-right; -1 ($FFFF) if the system direction is right-to-left.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Script.h`

IntlScript

Identifies the script system used by the Text Utilities date-formatting, time-formatting, and string-sorting functions. (Deprecated in Mac OS X v10.4. Use `ATSFontFamilyGetEncoding` instead.)

```
short IntlScript (
   void
);
```

Parameters

Return Value
A script code. All recognized script codes and their defined constants are listed in "Meta Script Codes" (page 1753). `IntlScript` returns only explicit script codes (□0). If the international resources selection flag is `TRUE`, the script code returned is that of the system script. If the identified script system is not enabled, the script code returned is that of the system script and the script-defaulted result flag is set to `TRUE`.

Discussion
Information about the script system is subject to two control flags—the font force flag and the international resources selection flag. You can test and set these flags with the `GetScriptManagerVariable` and `SetScriptManagerVariable` selectors `smFontForce` and `smIntlForce`.

The function starts by initializing two result flags, the script-forced result flag and the script-defaulted result flag, to `FALSE`. These flags are Script Manager variables, accessed through the `GetScriptManagerVariable` function selectors `smForced` and `smDefault`.

The function also identifies the script system whose resources are returned by the Script Manager function `GetIntlResource`. It is either the font script—the script system corresponding to the current font of the active graphics port—or the system script.

Special Considerations

`IntlScript` 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
`Script.h`

IntlTokenize

Allows your application to convert text into a sequence of language-independent tokens. (Deprecated in Mac OS X v10.4. There is no replacement because this function is no longer needed in Mac OS X.)

```
TokenResults IntlTokenize (
   TokenBlockPtr tokenParam
);
```

Parameters

tokenParam

A pointer to a token block structure. The structure specifies the text to be converted to tokens, the destination of the token list, a handle to the tokens (`'itl4'`) resource, and a set of options. See the `TokenBlock` (page 1736)data structure for information on what you need to pass in this structure and what you obtain on return.

Return Value
A `TokenResults` value that specifies whether the function executed with errors. See "Token Results" (page 1818) for a list of the values that can be returned.

Discussion
Before calling the `IntlTokenize` function, allocate memory for and set up the following data structures:

- A token block structure (data type `TokenBlock`). The token block structure is a parameter block that holds both input and output parameters for the `IntlTokenize` function.
- A token list to hold the results of the tokenizing operation. To set up the token list, estimate how many tokens will be generated from your text, multiply that by the size of a token structure, and allocate a memory block of that size in bytes. An upper limit to the possible number of tokens is the number of characters in the source text.
- A string list, if you want the `IntlTokenize` function to generate character strings for all the tokens. To set up the string list, multiply the estimated number of tokens by the expected average size of a string, and allocate a memory block of that size in bytes. An upper limit is twice the number of tokens plus the number of bytes in the source text.

The function `IntlTokenize` creates tokens based on information in the tokens (`'itl4'`) resource of the script system under which the source text was created. You must load the tokens resource and place its handle in the token block structure before calling the `IntlTokenize` function.

The token block structure contains both input and output values. At input, you must provide values for the fields that specify the source text location, the token list location, the size of the token list, the tokens (`'itl4'`) resource to use, and several options that affect the operation. You must set reserved locations to 0 before calling `IntlTokenize`.

On output, the token block structure specifies how many tokens have been generated and the size of the string list (if you have selected the option to generate strings).

The results of the tokenizing operation are contained in the token list, an array of token structures (data type `TokenRec` (page 1739)).

Pascal strings are generated if the `doString` parameter in the token block structure is set to `TRUE`. The string is a normalized version of the source text that generated the token; alternate digits are replaced with ASCII numerals, the decimal point is always an ASCII period, and double-byte Roman letters are replaced with low-ASCII equivalents.

To make a series of calls to `IntlTokenize` and append the results of each call to the results of previous calls, set `doAppend` to `FALSE` and initialize `tokenCount` and `stringCount` to 0 before making the first call to `IntlTokenize`. (You can ignore `stringCount` if you set `doString` to `FALSE`.) Upon completion of the call, `tokenCount` and `stringCount` will contain the number of tokens and the length in bytes of the string list, respectively, generated by the call. On subsequent calls, set `doAppend` to `TRUE`, reset the `source` and `sourceLength` parameters (and any other parameters as appropriate) for the new source text, but maintain the output values for `tokenCount` and `stringCount` from each call as input values to the next call. At the end of your sequence of calls, the token list and string list will contain, in order, all the tokens and strings generated from the calls to `IntlTokenize`.

If you are making tokens from text that was created under more than one script system, you must load the proper tokens resource and place its handle in the token block structure separately for each script run in the text, appending the results each time.

Delimiters for quoted literals are passed to `IntlTokenize` in a two-integer array.

The individual delimiters, as specified in the `leftDelims` and `rightDelims` parameters, are paired by position. The first (in storage order) opening delimiter in `leftDelims` is paired with the first closing delimiter in `rightDelims`.

Comment delimiters may be 1 or 2 tokens each and there may be two sets of opening and closing pairs. They are passed to `IntlTokenize` in a `commentType` array.

If only one token is needed for a delimiter, the second token must be specified to be `delimPad`. If only one delimiter of an opening-closing pair is needed, then both of the tokens allocated for the other symbol must be `delimPad`. The first token of a two-token sequence is at the higher position in the `leftComment` or `rightComment` array.

When `IntlTokenize` encounters an escape character within a quoted literal, it places the portion of the literal before the escape character into a single token (of type `tokenLiteral`), places the escape character into another token (`tokenEscape`), places the character following the escape character into another token (whatever token type it corresponds to), and places the portion of the literal following the escape sequence into another token (`tokenLiteral`). Outside of a quoted literal, the escape character has no special significance.

`IntlTokenize` considers the character specified in the `decimalCode` parameter to be a decimal character only when it is flanked by numeric or alternate numeric characters, or when it follows them.

Special Considerations

`IntlTokenize` may move memory; your application should not call this function at interrupt time.

Because each call to `IntlTokenize` must be for a single script run, there can be no change of script within a comment or quoted literal.

Comments and quoted literals must be complete within a single call to `IntlTokenize` in order to avoid syntax errors.

`IntlTokenize` always uses the tokens resource whose handle you pass it in the token block structure. Therefore, it is not directly affected by the state of the font force flag or the international resources selection flag. However, if you use the `GetIntlResource` function to get a handle to the tokens resource to pass to `IntlTokenize`, remember that `GetIntlResource` is affected by the state of the international resources selection flag.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Script.h`

SetScriptManagerVariable

Sets the specified Script Manager variable to the value of the input parameter. (Deprecated in Mac OS X v10.5. This is mainly used to set the value of variables that control the internal operation of the Script Manager (selectors `smIntlForce` and `smGenFlags`), and therefore there is no modern replacement.)

```
OSErr SetScriptManagerVariable (
   short selector,
   long param
);
```

Parameters

`selector`

A value that specifies a particular Script Manager variable. To specify the Script Manager variable whose value you wish to change, use one of the selector constants listed in "Script Manager Selectors" (page 1768).

`param`

The new value for the specified Script Manager variable.

The actual values to be assigned may be long integers, standard integers, or signed bytes. If the value is other than a long integer, you must store it in the low-order word or byte of the `param` parameter and set the unused bytes to 0.

Return Value
A result code. See "Script Manager Result Codes" (page 1821). The value `smBadVerb` if the selector is not valid. Otherwise, the function returns 0 (`noErr`).

Discussion
The Script Manager maintains a set of variables that control general settings of the text environment, including the identity of the system script and the keyboard script, and the settings of the font force flag and the international resources selection flag.

You may want access to the Script Manager variables in order to understand the current environment or to modify it.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Script.h`

SetScriptVariable

Sets the specified script variable for the specified script system to the value of the input parameter. (Deprecated in Mac OS X v10.5. The replacement for this function depends on the purpose for which it is used, as described in the Special Considerations section.)

```
OSErr SetScriptVariable (
   short script,
   short selector,
   long param
);
```

Parameters

script

A value that specifies the script system whose variable you are setting. Use one of the script-code constants listed in "Meta Script Codes" (page 1753).

selector

A value that specifies a particular script variable. Use one of the selector constants listed in "Script Variable Selectors" (page 1773).

param

The new value for the specified script variable. The actual value to be assigned may be a long integer, standard integer, or signed byte. If the value is not a long integer, you must store it in the low-order word or byte of the `param` parameter and set the unused bytes to 0.

Return Value
A result code. See "Script Manager Result Codes" (page 1821). The value `smBadVerb` if the selector is not valid, and `smBadScript` if the script is invalid. Otherwise, 0 (`noErr`).

Discussion
Each enabled script system maintains a set of variables that control the current settings of that script system, including the ID numbers of its international resources, its preferred fonts and font sizes, and its primary line direction.

Special Considerations

The replacement for this function depends on whether the goal is to set the keyboard layout globally or for a specific TSM document. To set it globally, use `TISSelectInputSource`. To set it for a specific document, use the TSM document property `kTSMDocumentInputSourceOverridePropertyTag`.

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
`Script.h`

SetSysDirection

Sets the value of `SysDirection`, the global variable that determines the system direction (primary line direction). (Deprecated in Mac OS X v10.4. There is no replacement because this function is no longer needed in Mac OS X.)

```
void SetSysDirection (
   short value
);
```

Parameters

value

The desired value for `SysDirection`:0 if you wish the system direction to be left-to-right and -1 ($FFFF) if you wish the system direction to be right-to-left.

Return Value

Discussion

The value of `SysDirection` is initialized from the system's international configuration resource, and may be controlled by the user. Your application can use the `SetSysDirection` function to change `SysDirection` while drawing, but should restore it when appropriate (such as when your application becomes inactive).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Script.h`

TransliterateText

Converts characters from one subscript to the closest possible approximation in a different subscript within the same double-byte script system. (Deprecated in Mac OS X v10.4. Use `CFStringUppercase` instead.)

```
OSErr TransliterateText (
   Handle srcHandle,
   Handle dstHandle,
   short target,
   long srcMask,
   ScriptCode script
);
```

Parameters

srcHandle

A handle to the source text to be transliterated. The `TransliterateText` function converts all of the text that you pass it in this parameter. It determines the length of the source text (in bytes) from the handle size.

dstHandle

A handle to a buffer that, upon completion of the call, contains the transliterated text.

Before calling `TransliterateText`, allocate a handle (of any size) to pass in the `dstHandle` parameter. The length of the transliterated text may be different (as when converting between single-byte and double-byte characters), and `TransliterateText` sets the size of the destination handle as required. It is your responsibility to dispose of the destination handle when you no longer need it.

target

A value that specifies what kind of text the source text is to be transliterated into.

The low byte of the target is the format to convert to (the target format). It determines what form the text should be transliterated to. In all script systems, there are two currently supported values for target format: `smTransAscii` and `smTransNative`. In double-byte script systems, additional values are recognized.

The high byte is the target modifier; it contains modifiers, whose meanings depend on the script code, providing additional formatting instructions. In all script systems, there are two values for target modifier: `smTransLower` and `smTransUpper`.

srcMask

A bit array that specifies which parts of the source text are to be transliterated. A bit is set for each script system or subscript that should be converted to the target format. In all script systems, the `srcMask` parameter may have the following values: `smMaskAscii`, `smMaskNative`, and `smMaskAll`. In double-byte script systems, additional values are recognized.

script

A value that specifies the script system of the text to be transliterated. Constants for all defined script codes are listed in "Meta Script Codes" (page 1753). To specify the font script, pass `smCurrentScript` in this parameter.

Return Value
A result code. See "Script Manager Result Codes" (page 1821).

Discussion
Transliteration is the conversion of text from one form or subscript to another within a single script system. In the Roman script system, transliteration means case conversion. In double-byte script systems, it is the automatic conversion of characters from one subscript to another. One common use for transliteration is as an initial stage of text conversion for an input method.

`TransliterateText` also performs uppercasing and lowercasing, with consideration for regional variants, in the Roman script system and on Roman text within double-byte script systems.

Because the low-ASCII character set (character codes $20–$7F) is present in all script systems, you could theoretically use the `TransliterateText` function to convert characters from one script system into another completely different script system. You could transliterate from a native subscript into ASCII under one script system, and then transliterate from that ASCII into a native subscript under a different script system. Such a function is not recommended, however, because of the imperfect nature of phonetic translation. Furthermore, many script systems do not support transliteration from native subscripts to ASCII.

Special Considerations

`TransliterateText` may move memory; your application should not call this function at interrupt time.

If you pass `smCurrentScript` in the `script` parameter, the conversion performed by `TransliterateText` can be affected by the state of the font force flag. It is unaffected by the international resources selection flag.

Transliteration of a block of text does not work across script-run boundaries. Because the `TransliterateText` function requires transliteration tables that are in a script system's international resources, you need to call it anew for each script run in your text.

Currently, the Roman version of `TransliterateText` checks the source mask only to ensure that at least one of the bits corresponding to the `smMaskAscii` and `smMaskNative` constants is set.

The Arabic and Hebrew versions of `TransliterateText` perform case conversion only. They allow the target values `smTransAscii` and `smTransNative` only; otherwise, they behave like the Roman version.

The `TransliterateText` tables for single-byte script systems reside in the script's string-manipulation (`'itl2'`) resource, so they can reflect region-specific or language-specific differences in uppercase conventions. If the string-manipulation resource does not include these tables, `TransliterateText` exits without doing anything.

The `TransliterateText` tables for double-byte script systems reside in the script's transliteration (`'trsl'`) resource. If the `'trsl'` resource does not include these tables, `TransliterateText` exits without doing anything.

The Japanese, Traditional Chinese, and Simplified Chinese versions of `TransliterateText` have two modes of operation. If either `smMaskAscii` or `smMaskNative` is specified in the source mask, and if the target is `smTransAscii`, and if either of the target modifiers is specified, `TransliterateText` performs the specified case conversion on both single-byte and double-byte Roman letters. Otherwise, `TransliterateText` performs conversions according to the target format values. Any combination of source masks and target format is permitted.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Script.h`

Data Types

CharByteTable

Represents an array of `char` values.

```
typedef  char CharByteTable[256];
```

Discussion
Used by the function `FillParseTable` (page 1720).

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

Declared In
`Script.h`

CommentType

Represents an array of `ScriptTokenType` values.

```
typedef ScriptTokenType CommentType[4];
```

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

Declared In
`Script.h`

DelimType

Represents an array of `ScriptTokenType` values.

```
typedef ScriptTokenType DelimType[2];
```

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

Declared In
`Script.h`

ScriptTokenType

Defins a data type for the script token type.

```
typedef  short ScriptTokenType;
```

Discussion

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

Declared In
`Script.h`

TokenBlock

Contains information about text that is to be converted to tokens, the destination of the token list, a handle to the tokens resource, and a set of options.

```
struct TokenBlock {
    Ptr source;
    long sourceLength;
    Ptr tokenList;
    long tokenLength;
    long tokenCount;
    Ptr stringList;
    long stringLength;
    long stringCount;
    Boolean doString;
    Boolean doAppend;
    Boolean doAlphanumeric;
    Boolean doNest;
    ScriptTokenType leftDelims[2];
    ScriptTokenType rightDelims[2];
    ScriptTokenType leftComment[4];
    ScriptTokenType rightComment[4];
    ScriptTokenType escapeCode;
    ScriptTokenType decimalCode;
    Handle itlResource;
    long reserved[8];
};
typedef struct TokenBlock TokenBlock;
typedef TokenBlock * TokenBlockPtr;
```

Fields

`source`
: A pointer to a stream of characters. On input to the function `IntlTokenize` (page 1729), a pointer to the beginning of the source text (not a Pascal string) to be converted.

`sourceLength`
: The length of the source stream. On input, the number of bytes in the source text.

`tokenList`
: A pointer to an array of tokens. On input, a pointer to a buffer you have allocated. On output, a pointer to a list of token structures generated by the `IntlTokenize` function.

`tokenLength`
: The maximum length of `TokenList`. On input, the maximum size of token list (in number of tokens, not bytes) that will fit into the buffer pointed to by the `tokenList` field.

`tokenCount`
: The number of tokens generated by the tokenizer. On input (if `doAppend = TRUE`), must contain the correct number of tokens currently in the token list. (Ignored if `doAppend = FALSE`.) On output, the number of tokens currently in the token list.

`stringList`
: A pointer to a stream of identifiers. On input (if `doString = TRUE`), a pointer to a buffer you have allocated. (Ignored if `doString = FALSE`) On output, a pointer to a list of strings generated by the `IntlTokenize` function.

`stringLength`
: The length of the string list. On input (if `doString = TRUE`), the size in bytes of the string list buffer pointed to by the `stringList` field. (Ignored if `doString = FALSE`.)

`stringCount`
: The number of bytes currently used. On input (if `doString = TRUE` and `doAppend = TRUE`), the correct current size in bytes of the string list. (Ignored if `doString = FALSE` or `doAppend = FALSE`.) On output, the current size in bytes of the string list. (Indeterminate if `doString = FALSE`.)

`doString`
: A Boolean value. On input, if `TRUE`, instructs `IntlTokenize` to create a Pascal string representing the contents of each token it generates. If `FALSE`, `IntlTokenize` generates a token list without an associated string list.

`doAppend`
: A Boolean value. On input, if `TRUE`, instructs `IntlTokenize` to append tokens and strings it generates to the current token list and string list. If `FALSE`, `IntlTokenize` writes over any previous contents of the buffer pointed to by `tokenList` and `stringList`.

`doAlphanumeric`
: A Boolean value. On input, if `TRUE`, instructs `IntlTokenize` to interpret numeric characters as alphabetic when mixed with alphabetic characters. If `FALSE`, all numeric characters are interpreted as numbers.

`doNest`
: A Boolean value. A value of type `Boolean`. On input, if `TRUE`, instructs `IntlTokenize` to allow nested comments (to any depth of nesting). If `FALSE`, comment delimiters may not be nested within other comment delimiters.

`leftDelims`
: A value of type `DelimType`. On input, an array of two integers, each of which contains the token code of the symbol that may be used as an opening delimiter for a quoted literal. If only one opening delimiter is needed, the other must be specified to be `delimPad`.

`rightDelims`
: A value of type `DelimType`. On input, an array of two integers, each of which contains the token code of the symbol that may be used as the matching closing delimiter for the corresponding opening delimiter in the `leftDelims` field.

`leftComment`
: A value of type `CommentType`. On input, an array of two pairs of integers, each pair of which contains codes for the two token types that may be used as opening delimiters for comments.

`rightComment`
: A value of type `CommentType`. On input, an array of two pairs of integers, each pair of which contains codes for the two token types that may be used as closing delimiters for comments.

`escapeCode`
: A value of type `TokenType`. On input, a single integer that contains the token code for the symbol that may be an escape character within a quoted literal.

`decimalCode`
: A value of type `TokenType`. On input, a single integer that contains the token type of the symbol to be used for a decimal point.

`itlResource`
: A value of type `Handle`. On input, a handle to the tokens (`'itl4'`) resource of the script system under which the source text was created.

`reserved`
: An 8-byte array of type `LongInt` . On input, this must be set to 0.

Discussion
The token block structure is a parameter block used to pass information to the `IntlTokenize` (page 1729) function and to retrieve results from it.

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

Declared In
Script.h

TokenRec

Contains information about the conversion of a sequence of characters to a token.

```
struct TokenRec {
    ScriptTokenType theToken;
    Ptr position;
    long length;
    StringPtr stringPosition;
};
typedef struct TokenRec TokenRec;
typedef TokenRec * TokenRecPtr;
```

Fields

`theToken`
A numeric code that specifies the type of token (such as whitespace, opening parenthesis, alphabetic or numeric sequence) described by this token structure. Constants for all defined token codes are listed in "Obsolete Token Codes" (page 1821).

`position`
A pointer to the first character in the source text that caused this particular token to be generated.

`length`
The length, in bytes, of the source text that caused this particular token to be generated.

`stringPosition`
If `doString` = `TRUE`, a pointer to a null-terminated Pascal string, padded if necessary so that its total number of bytes (length byte + text + null byte + padding) is even. If `doString` = `FALSE`, this field is `NULL`.

The value in the length byte of the null-terminated Pascal string does not include either the terminating zero byte or the possible additional padding byte. There may be as many as two additional bytes beyond the specified length.

Discussion
The token structure holds the results of the conversion of a sequence of characters to a token by the `IntlTokenize` (page 1729) function. When it analyzes text, `IntlTokenize` generates a token list, which is a sequence of token structures.

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

Declared In
Script.h

Constants

Assorted Constants

Calendar Codes

Specify constants for various calendars.

```
enum {
    calGregorian = 0,
    calArabicCivil = 1,
    calArabicLunar = 2,
    calJapanese = 3,
    calJewish = 4,
    calCoptic = 5,
    calPersian = 6
};
```

Constants

`calGregorian`
Specifies the Gregorian calendar.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`calArabicCivil`
Specifies the Arabic civil calendar.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`calArabicLunar`
Specifies the Arabic lunar calendar.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`calJapanese`
Specifies the Japanese calendar.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`calJewish`
Specifies the Jewish calendar.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`calCoptic`
Specifies the Coptic calendar.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`calPersian`
Specifies the Persian calendar.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Discussion
These calendar codes are bit numbers, not masks.

Character Byte Types

Specify character byte types.

```
enum {
    smSingleByte = 0,
    smFirstByte = -1,
    smLastByte = 1,
    smMiddleByte = 2
};
```

Constants
`smSingleByte`
Specifes a single byte.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smFirstByte`
Specifies the first byte.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smLastByte`
Specifies the last byte.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smMiddleByte`
Specifies the middle byte.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Character Types

Specify basic character types.

```
enum {
    smCharPunct = 0x0000,
    smCharAscii = 0x0001,
    smCharEuro = 0x0007,
    smCharExtAscii = 0x0007,
    smCharKatakana = 0x0002,
    smCharHiragana = 0x0003,
    smCharIdeographic = 0x0004,
    smCharTwoByteGreek = 0x0005,
    smCharTwoByteRussian = 0x0006,
    smCharBidirect = 0x0008,
    smCharContextualLR = 0x0009,
    smCharNonContextualLR = 0x000A,
    smCharHangul = 0x000C,
    smCharJamo = 0x000D,
    smCharBopomofo = 0x000E,
    smCharGanaKana = 0x000F,
    smCharFISKana = 0x0002,
    smCharFISGana = 0x0003,
    smCharFISIdeo = 0x0004
};
```

Constants

`smCharPunct`
Specifies punctuation characters.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharAscii`
Specifies ASCII characters.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharEuro`
Specifies `smCharEuro`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharExtAscii`
Specifies a more correct synonym for `smCharEuro`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharKatakana`
Specifies additional character types for Japanese Katakana.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharHiragana`
Specifies additional character types for Japanese Hiragana.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharIdeographic`
Specifies additional character types for Hanzi, Kanji, and Hanja.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharTwoByteGreek`
Specifies additional character types for double-byte Greek in Far East systems.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharTwoByteRussian`
Specifies additional character types for double-byte Cyrillic in Far East systems.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharBidirect`
Specifies additional character types for Arabic/Hebrew.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharContextualLR`
Specifies contextual left-right: Thai, Indic scripts.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharNonContextualLR`
Specifies additional character types for non-contextual left-right: Cyrillic, Greek.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharHangul`
Specifies additional character types for Korean Hangul.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharJamo`
Specifies additional character types for Korean Jamo.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharBopomofo`
Specifies additional character types for Chinese Bopomofo.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharGanaKana`
Specifies additional character types shared for Japanese Hiragana and Katakana.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharFISKana`
Specifies obsolete Katakana names, for backward compatibility.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smCharFISGana`
Specifies obsolete Hiragana namde, for backward compatibility.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smCharFISIdeo`
Specifies obsolete Hanzi, Kanji, and Hanja names, for backward compatibility.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Character Type Classes

Specify character-type classes for double-byte script systems.

```
enum {
    smCharFISGreek = 0x0005,
    smCharFISRussian = 0x0006,
    smPunctNormal = 0x0000,
    smPunctNumber = 0x0100,
    smPunctSymbol = 0x0200,
    smPunctBlank = 0x0300,
    smPunctRepeat = 0x0400,
    smPunctGraphic = 0x0500,
    smKanaSmall = 0x0100,
    smKanaHardOK = 0x0200,
    smKanaSoftOK = 0x0300,
    smIdeographicLevel1 = 0x0000,
    smIdeographicLevel2 = 0x0100,
    smIdeographicUser = 0x0200,
    smFISClassLvl1 = 0x0000,
    smFISClassLvl2 = 0x0100,
    smFISClassUser = 0x0200,
    smJamoJaeum = 0x0000,
    smJamoBogJaeum = 0x0100,
    smJamoMoeum = 0x0200,
    smJamoBogMoeum = 0x0300
};
```

Constants

`smCharFISGreek`
Specfies character-type classes for double-byte Greek in Far East systems.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smCharFISRussian`
Specfies character-type classes for double-byte Cyrillic in Far East systems.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smPunctNormal`
Specfies character-type classes for normal punctuation (`smCharPunct`).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smPunctNumber`
Specfies character-type classes for number punctuation (`smCharPunct`).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smPunctSymbol`
Specfies character-type classes for symbol punctuation (`smCharPunct`).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smPunctBlank`
Specfies additional character-type classes for punctuation in double-byte systems.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smPunctRepeat`
Specifies a character-type class for repeat markers.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smPunctGraphic`
Specifies a character-type class forl ine graphics.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKanaSmall`
Specfies character-type classes for Katakana and Hiragana double-byte systems.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKanaHardOK`
Specfies character-type classes for Katakana and Hiragana double-byte systems; can have dakuten.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKanaSoftOK`
Specfies character-type classes for Katakana and Hiragana double-byte systems; can have dakuten or han-dakuten.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smIdeographicLevel1`
Specfies character-type classes for Ideographic double-byte systems; level 1 char.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smIdeographicLevel2`
Specfies character-type classes for Ideographic double-byte systems; level 2 char.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smIdeographicUser`
Specfies character-type classes for Ideographic double-byte systems; user char.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smFISClassLvl1`
Obsolete, for backward compatibility; level 1 char.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smFISClassLvl2`
Obsolete, for backward compatibility; level 2 char.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smFISClassUser`
Obsolete, for backward compatibility; user char.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smJamoJaeum`
Specfies character-type Jamo classes for Korean systems; simple consonant char.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smJamoBogJaeum`
Specfies character-type Jamo classes for Korean systems; complex consonant char.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smJamoMoeum`
Specfies character-type Jamo classes for Korean systems; simple vowel char.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smJamoBogMoeum`
Specfies character-type Jamo classes for Korean systems; complex vowel char.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

Character Type Field Masks

Specify masks used to extract information from the return value of the `CharacterType` function.

```
enum {
    smcTypeMask = 0x000F,
    smcReserved = 0x00F0,
    smcClassMask = 0x0F00,
    smcOrientationMask = 0x1000,
    smcRightMask = 0x2000,
    smcUpperMask = 0x4000,
    smcDoubleMask = 0x8000
};
```

Constants

`smcTypeMask`
Character-type mask.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smcReserved`
Reserved.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smcClassMask`
Character-class mask.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smcOrientationMask`
Character orientation (double-byte scripts).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smcRightMask`
Writing direction (bidirectional scripts); main character set or subset (double-byte scripts)
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smcUpperMask`
Uppercase or lowercase.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smcDoubleMask`
Size (1 or 2 bytes).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

Discussion

These bit masks are used to extract fields from the return value of the `CharacterType` (page 1717) function.

The character type of the character in question is the result of performing an `AND` operation with `smcTypeMask` and the `CharacterType` result.

The character class of the character in question is the result of performing an `AND` operation with `smcClassMask` and the `CharacterType` result. Character classes can be considered as subtypes of character types.

The orientation of the character in question is the result of performing an `AND` operation with `smcOrientationMask` and the `CharacterType` result. The orientation value can be either `smCharHorizontal` or `smCharVertical`.

The direction of the character in question is the result of performing an `AND` operation with `smcRightMask` and the `CharacterType` result. The direction value can be either `smCharLeft` (left-to-right) or `smCharRight` (right-to-left).

The case of the character in question is the result of performing an `AND` operation with `smcUpperMask` and the `CharacterType` result. The case value can be either `smCharLower` or `smCharUpper`.

The size of the character in question is the result of performing an `AND` operation with `smcDoubleMask` and the `CharacterType` result. The size value can be either `smChar1byte` or `smChar2byte`.

Character Set Extensions

Specify extensions to character sets.

```
enum {
    diaeresisUprY = 0xD9,
    fraction = 0xDA,
    intlCurrency = 0xDB,
    leftSingGuillemet = 0xDC,
    rightSingGuillemet = 0xDD,
    fiLigature = 0xDE,
    flLigature = 0xDF,
    dblDagger = 0xE0,
    centeredDot = 0xE1,
    baseSingQuote = 0xE2,
    baseDblQuote = 0xE3,
    perThousand = 0xE4,
    circumflexUprA = 0xE5,
    circumflexUprE = 0xE6,
    acuteUprA = 0xE7,
    diaeresisUprE = 0xE8,
    graveUprE = 0xE9,
    acuteUprI = 0xEA,
    circumflexUprI = 0xEB,
    diaeresisUprI = 0xEC,
    graveUprI = 0xED,
    acuteUprO = 0xEE,
    circumflexUprO = 0xEF,
    appleLogo = 0xF0,
    graveUprO = 0xF1,
    acuteUprU = 0xF2,
    circumflexUprU = 0xF3,
    graveUprU = 0xF4,
    dotlessLwrI = 0xF5,
    circumflex = 0xF6,
    tilde = 0xF7,
    macron = 0xF8,
    breveMark = 0xF9,
    overDot = 0xFA,
    ringMark = 0xFB,
    cedilla = 0xFC,
    doubleAcute = 0xFD,
    ogonek = 0xFE,
    hachek = 0xFF
};
```

Keyboard Script Synchronization

Specifies to disable font and keyboard script synchronization.

```
enum {
    smfDisableKeyScriptSync = 27
};
```

Glyph Orientations

Specify character-type glyph orientation for double-byte systems.

```
enum {
    smCharHorizontal = 0x0000,
    smCharVertical = 0x1000,
    smCharLeft = 0x0000,
    smCharRight = 0x2000,
    smCharLower = 0x0000,
    smCharUpper = 0x4000,
    smChar1byte = 0x0000,
    smChar2byte = 0x8000
};
```

Constants

`smCharHorizontal`
Specifies horizontal character form.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharVertical`
Specifies vertical character form.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharLeft`
Specifies left character direction.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharRight`
Specifies right character direction.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharLower`
Specifies lowercase character modifers.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCharUpper`
Specifies uppercase character modifers.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smChar1byte`
Specifies character size modifiers (single or multiple bytes).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smChar2byte`
Specifies character size modifiers (single or multiple bytes).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

Keyboard Script Switching Selectors

Specify a keyboard script switching flag and mask.

```
enum {
    smKeyForceKeyScriptBit = 7,
    smKeyForceKeyScriptMask = 1 << smKeyForceKeyScriptBit
};
```

Constants

`smKeyForceKeyScriptBit`

A flag that specifies to force keyboard script switching.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeyForceKeyScriptMask`

A mask that specifies to force keyboard script switching.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Keyboard Script Values

Specify actions for keyboard scripts.

```
enum {
    smKeyNextScript = -1,
    smKeySysScript = -2,
    smKeySwapScript = -3,
    smKeyNextKybd = -4,
    smKeySwapKybd = -5,
    smKeyDisableKybds = -6,
    smKeyEnableKybds = -7,
    smKeyToggleInline = -8,
    smKeyToggleDirection = -9,
    smKeyNextInputMethod = -10,
    smKeySwapInputMethod = -11,
    smKeyDisableKybdSwitch = -12,
    smKeySetDirLeftRight = -15,
    smKeySetDirRightLeft = -16,
    smKeyRoman = -17
};
```

Constants

`smKeyNextScript`

Specifies to switch to the next available script.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeySysScript`

Specfiies to switch to the system script.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeySwapScript`
Specifies to switch to the previously-used script

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeyNextKybd`
Specifies to switch to the next keyboard in current keyscript.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeySwapKybd`
Specfies to switch to a previously-used keyboard in the current keyscript.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeyDisableKybds`
Specifies to disable keyboards not in the system or Roman script.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeyEnableKybds`
Specifies to enable keyboards for all enabled scripts.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeyToggleInline`
Specifies to toggle inline input for the current keyscript

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeyToggleDirection`
Specifies to toggle the default line direction (`TESysJust`).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeyNextInputMethod`
Specfies to switch to the next input method in the current keyscript.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeySwapInputMethod`
Specfies to switch to the last-used input method in the current keyscript.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeyDisableKybdSwitch`
Specfies to disable switching from the current keyboard.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeySetDirLeftRight`
Specfies to set the default line direction to left-right, align left.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeySetDirRightLeft`
Specfies to set the default line direction to right-left, align right.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeyRoman`
Specfies to set the keyscript to Roman. Does nothing if on a Roman-only system. This is unlike `KeyScript(smRoman)` which forces an update to current default Roman keyboard. See `KeyScript` documentation for more information.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Keyboard Synchronization Mask

Disables font and keyboard script synchronization mask

```
enum {
    smfDisableKeyScriptSyncMask = 1L << smfDisableKeyScriptSync
};
```

Constants

`smfDisableKeyScriptSyncMask`
Disable font and keyboard script synchronization mask

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Discussion

Meta Script Codes

Specify implicit script codes.

```
enum {
    smSystemScript = -1,
    smCurrentScript = -2,
    smAllScripts = -3
};
```

Constants

`smSystemScript`
Specifies the system script.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smCurrentScript`
Specifies the font script.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smAllScripts`
Specfies any script.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Discussion
You can specify script systems with implicit and explicit script code constants in the `script` parameter of the `GetScriptVariable` (page 1726) and `SetScriptVariable` (page 1732) functions. The implicit script codes `smSystemScript` and `smCurrentScript` are special negative values for the system script and the font script, respectively.

Negative Verbs

Specify special negative verbs that were associated with WorldScript I.

```
enum {
    smLayoutCache = -309,
    smOldVerbSupport = -311,
    smSetKashidas = -291,
    smSetKashProp = -287,
    smScriptSysBase = -281,
    smScriptAppBase = -283,
    smScriptFntBase = -285,
    smScriptLigatures = -263,
    smScriptNumbers = -267
};
```

Constants
`smLayoutCache`
Specifies that `HiWrd(param)` is the number of entries, `LoWrd` is the maximum input length

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smOldVerbSupport`
Specifies that a parameter is added to old verbs to map to WorldScript I verb.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smSetKashidas`
Specifies parameter is on or off; obsolete verb = -36.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smSetKashProp`
Specifies parameter is kashida proportion; obsolete verb = -32.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Script.h`.

`smScriptSysBase`
Specifies parameter is associated font to use with the system font; obsolete verb = -26)
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Script.h`.

`smScriptAppBase`
Specifies parameter is associated font to use with application font; obsolete verb = -28.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Script.h`.

`smScriptFntBase`
Specifies that a parameter is associated font to use with all other fonts; obsolete verb = -30.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Script.h`.

`smScriptLigatures`
Obsolete verb = -8.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Script.h`.

`smScriptNumbers`
Obsolete verb = -12.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Script.h`.

Numeral Codes

Specify the kinds of numerals used by a script.

```
enum {
    intWestern = 0,
    intArabic = 1,
    intRoman = 2,
    intJapanese = 3,
    intEuropean = 4,
    intOutputMask = 0x8000
};
```

Constants

`intWestern`
Specifies Western numerals.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`intArabic`
Specifies Native Arabic numerals.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`intRoman`
Specifies Roman numerals.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`intJapanese`
Specifies Japanese numerals.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`intEuropean`
Specifies European numerals.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`intOutputMask`
Specifies an output mask.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

Discussion
These constants specify bit numbers, not masks.

Script Redraw Selectors

Specify values for script redraw flags.

```
enum {
    smRedrawChar = 0,
    smRedrawWord = 1,
    smRedrawLine = -1
};
```

Constants

`smRedrawChar`

Specifies to redraw character only.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smRedrawWord`

Specifies to redraw entire word (double-byte systems).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smRedrawLine`

Specifies to redraw entire line (bidirectional systems).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

Script Codes

Specify Mac OS encodings that are related to a `FOND` ID range.

```
enum {
    smRoman = 0,
    smJapanese = 1,
    smTradChinese = 2,
    smKorean = 3,
    smArabic = 4,
    smHebrew = 5,
    smGreek = 6,
    smCyrillic = 7,
    smRSymbol = 8,
    smDevanagari = 9,
    smGurmukhi = 10,
    smGujarati = 11,
    smOriya = 12,
    smBengali = 13,
    smTamil = 14,
    smTelugu = 15,
    smKannada = 16,
    smMalayalam = 17,
    smSinhalese = 18,
    smBurmese = 19,
    smKhmer = 20,
    smThai = 21,
    smLao = 22,
    smGeorgian = 23,
    smArmenian = 24,
    smSimpChinese = 25,
    smTibetan = 26,
    smMongolian = 27,
    smEthiopic = 28,
    smGeez = 28,
    smCentralEuroRoman = 29,
    smVietnamese = 30,
    smExtArabic = 31,
    smUninterp = 32
};
```

Constants

`smRoman`

Specifies the Roman script system.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smJapanese`

Specifies the Japanese script system.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smTradChinese`

Specifies the traditional Chinese script system.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKorean`
Specifies the Korean script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smArabic`
Specifies the Arabic script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smHebrew`
Specifies the Hebrew script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smGreek`
Specifies the Greek script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCyrillic`
Specifies the Cyrillic script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smRSymbol`
Specifies right-to-left symbols. The script code represented by the constant `smRSymbol` is available as an alternative to `smUninterp`, for representation of special symbols that have a right-to-left line direction. Note, however, that the script management system provides no direct support for representation of text with this script code.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smDevanagari`
Specifies the Devanagari script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smGurmukhi`
Specifies the Gurmukhi script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smGujarati`
Specifies the Gujarati script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smOriya`
Specifies the Oriya script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smBengali`
Specifies the Bengali script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smTamil`
Specifies the Tamil script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smTelugu`
Specifies the Telugu script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smKannada`
Specifies the Kannada/Kanarese script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smMalayalam`
Specifies the Malayalam script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smSinhalese`
Specifies the Sinhalese script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smBurmese`
Specifies the Burmese script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smKhmer`
Specifies the Khmer script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smThai`
Specifies the Thai script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smLao`
Specifies the Laotian script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smGeorgian`
Specifies the Georgian script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smArmenian`
Specifies the Armenian script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smSimpChinese`
Specifies the simplified Chinese script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smTibetan`
Specifies the Tibetan script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smMongolian`
Specifies the Mongolian script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smEthiopic`
Specifies the Geez/Ethiopic script system. This constant is the same as `smGeez`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smGeez`
Specifies the Geez/Ethiopic script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCentralEuroRoman`
Used for Czech, Slovak, Polish, Hungarian, Baltic languages.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smVietnamese`
Specifies the Extended Roman script system for Vietnamese.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smExtArabic`
Specifies the extended Arabic for Sindhi script system.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smUninterp`
: Uninterpreted symbols. The script code represented by the constant `smUninterp` is available for representation of special symbols, such as items in a tool palette, that must not be considered as part of any actual script system. For manipulating and drawing such symbols, the `smUninterp` constant should be treated as if it indicated the Roman script system rather than the system script; that is, the default behavior of uninterpreted symbols should be Roman.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Script Code - Unicode Input

Specifies the extended script code for full Unicode input.

```
enum {
    smUnicodeScript = 0x7E
};
```

Script Constants

Specify constants used to get and set script variables.

```
enum {
    smScriptNumDate = 30,
    smScriptKeys = 32,
    smScriptIcon = 34,
    smScriptPrint = 36,
    smScriptTrap = 38,
    smScriptCreator = 40,
    smScriptFile = 42,
    smScriptName = 44,
    smScriptMonoFondSize = 78,
    smScriptPrefFondSize = 80,
    smScriptSmallFondSize = 82,
    smScriptSysFondSize = 84,
    smScriptAppFondSize = 86,
    smScriptHelpFondSize = 88,
    smScriptValidStyles = 90,
    smScriptAliasStyle = 92
};
```

Constants

`smScriptNumDate`
: (2 bytes) The numeral code and calendar code for the script. The numeral code specifies the kind of numerals the script uses, and is in the high-order byte of the word the calendar code specifies the type of calendar it uses and is in the low-order byte of the word. The value of this variable is initialized from the script system's international bundle resource. It may be changed during execution when the user selects, for example, a new calendar from a script system's control panel. See "Numeral Codes" (page 1755) and "Calendar Codes" (page 1740) for the different codes.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptKeys`
(2 bytes) The resource ID of the script's current keyboard-layout (`'KCHR'`) resource. The keyboard-layout resource is used to map virtual key codes into the correct character codes for the script. The value of this variable is initialized from the script system's international bundle resource. It is updated when the user selects a new keyboard layout, or when the application calls the `KeyScript` function. You can force a particular keyboard layout to be used with your application by setting the value of this variable and then calling `KeyScript`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptIcon`
(2 bytes) The resource ID of the script's keyboard icon family (resource types `'kcs#'`, `'kcs4'`, and `'kcs8'`). The keyboard icon family consists of the keyboard icons displayed in the keyboard menu. The value of this variable is initialized from the script system's international bundle resource. Note that, unlike `smScriptKeys`, the value of this variable is not automatically updated when the keyboard layout changes. (System software assumes that the icon family has an identical ID to the keyboard-layout resource, and usually ignores this variable.)

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptPrint`
(4 bytes) The print action function vector, set up by the script system (or by the Script Manager if the `smsfAutoInit` bit is set) when the script is initialized.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptTrap`
(4 bytes) A pointer to the script's script-structure dispatch function (for internal use only).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptCreator`
(4 bytes) The 4-character creator type for the script system's file, that is, the file containing the script system. For the Roman script system, it is `'ZSYS'`, for WorldScript I it is `'univ'`, and for World Script II it is `'doub'`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptFile`
(4 bytes) A pointer to the Pascal string that contains the name of the script system's file, that is, the file containing the script system. For the Roman script system, the string is `'System'`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptName`

(4 bytes) A pointer to a Pascal string that contains the script system's name. For the Roman script system and single-byte simple script systems, the string is `'Roman'`. For single-byte complex script systems, this name is taken from the encoding/rendering (`'itl5'`) resource. For double-byte script systems, it is taken from the WorldScript II extension and is `'WorldScript II'`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptMonoFondSize`

(4 bytes) The default font family ID and size (in points) for monospaced text. The ID is stored in the high-order word, and the size is stored in the low-order word. The value of this variable is taken from the script system's international bundle resource. Note that not all script systems have a monospaced font.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptPrefFondSize`

(4 bytes) Currently not used.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptSmallFondSize`

(4 bytes) The default font family ID and size (in points) for small text, generally the smallest font and size combination that is legible on screen. The ID is stored in the high-order word, and the size is stored in the low-order word. Sizes are important for example, a 9-point font may be too small in Chinese. The value of this variable is taken from the script system's international bundle resource.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptSysFondSize`

(4 bytes) The default font family ID and size (in points) for this script system's preferred system font. The ID is stored in the high-order word, and the size is stored in the low-order word. The value of this variable is taken from the script system's international bundle resource.

This variable holds similar information to the variable accessed through the `smScriptSysFond` selector. If you need font family ID only and don't want size information, it is simpler to use `smScriptSysFond`. Note, however, that changing the value of this variable has no effect on the value accessed through `smScriptSysFond`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptAppFondSize`
: (4 bytes) The default font family ID and size (in points) for this script system's preferred application font. The ID is stored in the high-order word, and the size is stored in the low-order word. The value of this variable is taken from the script system's international bundle resource.

  This variable holds similar information to the variable accessed through the `smScriptAppFond` selector. If you need font family ID only and don't want size information, it is simpler to use `smScriptAppFond`. Note, however, that changing the value of this variable has no effect on the value accessed through `smScriptAppFond`.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptHelpFondSize`
: (4 bytes) The default font family ID and size (in points) for Balloon Help. The ID is stored in the high-order word, and the size is stored in the low-order word. Sizes are important for example, a 9-point font may be too small in Chinese. The value of this variable is taken from the script system's international bundle resource.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptValidStyles`
: (1 byte) The set of all valid styles for the script. For example, the Extended style is not valid in the Arabic script. When the `GetScriptVariable` function is called with the `smScriptValidStyles` selector, the low-order byte of the returned value is a style code that includes all of the valid styles for the script (that is, the bit corresponding to each QuickDraw style is set if that style is valid for the specified script). The value of this variable is taken from the script system's international bundle resource.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptAliasStyle`
: (1 byte) The style to use for indicating aliases. When the `GetScriptVariable` function is called with `smScriptAliasStyle`, the low-order byte of the returned value is the style code that should be used in that script for indicating alias names (for example, in the Roman script system, alias names are indicated in italics). The value of this variable is taken from the script system's international bundle resource.

  Some script systems, such as Arabic and Hebrew, have private script-system selectors that are unique to those scripts. Those private selectors are negative, whereas selectors that extend across script systems are positive.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

Script Flag Attributes

Specify bits used to examine attributes in the script flags word.

```
enum {
    smsfIntellCP = 0,
    smsfSingByte = 1,
    smsfNatCase = 2,
    smsfContext = 3,
    smsfNoForceFont = 4,
    smsfBODigits = 5,
    smsfAutoInit = 6,
    smsfUnivExt = 7,
    smsfSynchUnstyledTE = 8,
    smsfForms = 13,
    smsfLigatures = 14,
    smsfReverse = 15,
    smfShowIcon = 31,
    smfDualCaret = 30,
    smfNameTagEnab = 29,
    smfUseAssocFontInfo = 28
};
```

Constants

`smsfIntellCP`
Specifies the script can support intelligent cut and paste (it uses spaces as word delimiters).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smsfSingByte`
Specifies the script has only single-byte characters.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smsfNatCase`
Specifies the script has both uppercase and lowercase native characters.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smsfContext`
Specifies the script is contextual.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smsfNoForceFont`
Specifies the script does not support font forcing (ignores the font force flag).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smsfBODigits`
Specifies the script has alternate digits at $B0–$B9. Arabic and Hebrew, for example, have their native numeric forms at this location in their character sets.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smsfAutoInit`
Specifies the script is initialized by the Script Manager. Single-byte simple script systems can set this bit to avoid having to initialize themselves.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smsfUnivExt`
Specifies the script uses the WorldScript I extension.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smsfSynchUnstyledTE`
Specifies the script synchronizes keyboard with font for monostyled TextEdit.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smsfForms`
Specifies to use contextual forms if this bit is set; do not use them if it is cleared.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smsfLigatures`
Specifies to use contextual ligatures if this bit is set; do not use them if it is cleared.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smsfReverse`
Specifies reverse right-to-left text to draw it in (left-to-right) display order if this bit is set; do not reorder text if this bit is cleared.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smfShowIcon`
Specifies to show icon even if only one script; bits in the `smGenFlags` long.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smfDualCaret`
Specifies to use dual caret for mixed direction text; bits in the `smGenFlags` long.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smfNameTagEnab`
Reserved for internal use; bits in the `smGenFlags` long.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smfUseAssocFontInfo`
Specifies to set the associated font info for `FontMetrics` calls; bits in the `smGenFlags` long.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Discussion
These constants are available for examining attributes in the script flags word. Bits above 8 are nonstatic, meaning that they may change during program execution. (Note that the constant values represent bit numbers in the flags word, not masks.)

Script Manager Selectors

Specify selectors you can use with the functions `GetScriptManagerVariable` and `SetScriptManagerVariable`.

```
enum {
    smVersion = 0,
    smMunged = 2,
    smEnabled = 4,
    smBidirect = 6,
    smFontForce = 8,
    smIntlForce = 10,
    smForced = 12,
    smDefault = 14,
    smPrint = 16,
    smSysScript = 18,
    smLastScript = 20,
    smKeyScript = 22,
    smSysRef = 24,
    smKeyCache = 26,
    smKeySwap = 28,
    smGenFlags = 30,
    smOverride = 32,
    smCharPortion = 34,
    smDoubleByte = 36,
    smKCHRCache = 38,
    smRegionCode = 40,
    smKeyDisableState = 42
};
```

Constants
`smVersion`

The Script Manager version number (2 bytes) . This variable has the same format as the version number obtained from calling the `Gestalt` function with the Gestalt selector `gestaltScriptMgrVersion`. The high-order byte contains the major version number, and the low-order byte contains the minor version number.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smMunged`

The modification count for Script Manager variables (2 bytes) . At startup, `smMunged` is initialized to 0, and it is incremented when the `KeyScript` function changes the current keyboard script and updates the variables accessed via `smKeyScript` and `smLastScript`. The `smMunged` selector is also incremented when the `SetScriptManagerVariable` function is used to change a Script Manager variable. You can check this variable at any time to see whether any of your own data structures that may depend on Script Manager variables need to be updated.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smEnabled`

The script count (1 byte) ; the number of currently enabled script systems. At startup time, the Script Manager initializes the script count to 0, then increments it for each installed and enabled script system (including Roman). You can use `smEnabled` to determine whether more than one script system is installed—that is, whether your application needs to handle non-Roman text.

Never call `SetScriptManagerVariable` with the `smEnabled` selector. It could result in inconsistency with other script system values.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smBidirect`

The bidirectional flag, which indicates when at least one bidirectional script system is enabled. This flag is set to `TRUE` ($FF) if the Arabic or Hebrew script system is enabled.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smFontForce`

The font force flag (1 byte). At startup, the Script Manager sets its value from the system script's international configuration (`'itlc'`) resource. The flag returns 0 for `FALSE` and $FF for `TRUE`. If the system script is non-Roman, the font force flag controls whether a font with ID in the Roman script range is interpreted as belonging to the Roman script or to the system script.

When you call `SetScriptManagerVariable` with the `smFontForce` selector, be sure to pass only the value 0 or $FF, or a later call to `GetScriptManagerVariable` may return an unrecognized value.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smIntlForce`

The international resources selection flag (1 byte). At startup, the Script Manager sets its value from the system script's international configuration (`'itlc'`) resource. The flag returns 0 for `FALSE` and `$FF` for `TRUE`. This flag controls whether international resources of the font script or the system script are used for string manipulation.

When you call `SetScriptManagerVariable` with the `smIntlForce` selector, be sure to pass only the value 0 or $FF, or a later call to `GetScriptManagerVariable` may return an unrecognized value.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smForced`

The script-forced result flag (1 byte). If the current script has been forced to the system script, this flag is set to `TRUE`. Use the `smForced` selector to obtain reports of the actions of the `FontScript`, `FontToScript`, and `IntlScript` functions. This variable is for information only; never set its value with `SetScriptManagerVariable`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smDefault`

The script-defaulted result flag (1 byte). If the script system corresponding to a specified font is not available, this flag is set to `TRUE`. Use this selector to obtain reports of the actions of the `FontScript`, `FontToScript`, and `IntlScript` functions. This variable is for information only; never set its value with `SetScriptManagerVariable`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smPrint`

The print action function vector, set up by the Script Manager at startup (4 bytes).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smSysScript`

The system script code (2 bytes) . At startup, the Script Manager initializes this variable from the system script's international configuration (`'itlc'`) resource. This variable is for information only; never set its value with `SetScriptManagerVariable`. Constants for all defined script codes are listed in "Region Codes A" (page 1798).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smLastScript`

The previously used keyboard script (2 bytes). When you change keyboard scripts with the `KeyScript` function, the Script Manager moves the old value of `smKeyScript` into `smLastScript`. `KeyScript` can also swap the current keyboard script with the previous keyboard script, in which case the contents of `smLastScript` and `smKeyScript` are swapped. Constants for all defined script codes are listed in "Region Codes A" (page 1798). Never set the value of this variable with `SetScriptManagerVariable`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smKeyScript`

The current keyboard script (2 bytes) . The `KeyScript` function tests and updates this variable. When you change keyboard scripts with the `KeyScript` function, the Script Manager moves the old value of `smKeyScript` into `smLastScript`. `KeyScript` can also swap the current keyboard script with the previous keyboard script, in which case the contents of `smLastScript` and `smKeyScript` are swapped. The Script Manager also uses this variable to get the proper keyboard icon and to retrieve the proper keyboard-layout (`'KCHR'`) resource. Constants for all defined script codes are listed in "Region Codes A" (page 1798). Never set the value of this variable directly with `SetScriptManagerVariable`; call `KeyScript` to change keyboard scripts.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smSysRef`

The System Folder volume reference number (2 bytes) . Its value is initialized from the system global variable `BootDrive` at startup.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smKeyCache`

An obsolete variable (4 bytes). This variable at one time held a pointer to the keyboard cache. The value it provided was not correct and should not be used.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smKeySwap`

A handle to the keyboard-swap (`'KSWP'`) resource (4 bytes). The Script Manager initializes the handle at startup. The keyboard-swap resource controls the key combinations with which the user can invoke various actions with the `KeyScript` function, such as switching among script systems.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smGenFlags`

The general flags used by the Script Manager (4 bytes). The Script Manager general flags is a long word value its high-order byte is set from the flags byte in the system script's international configuration (`'itlc'`) resource. These constants are available to designate bits in the variable accessed through `smGenFlags`:

- `smfNameTagEnab` (a value of 29)Reserved for internal use.
- `smfDualCaret` (a value of 30)Use a dual caret for mixed-directional text.
- `smfShowIcon` (a value of 31)Show the keyboard menu even if only one keyboard layout or one script (Roman) is available. (This bit is checked only at system startup.)

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smOverride`

The script override flags (4 bytes). At present, these flags are not set or used by the Script Manager. They are, however, reserved for future use.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smCharPortion`

A value used by script systems to allocate intercharacter and interword spacing when justifying text (2 bytes). It denotes the weight allocated to intercharacter space versus interword space. The value of this variable is initialized to 10 percent by the Script Manager, although it currently has no effect on text of the Roman script system. The variable is in 4.12 fixed-point format, which is a 16-bit signed number with 4 bits of integer and 12 bits of fraction. (In that format, 10 percent has the hexadecimal value $0199.)

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smDoubleByte`

The double-byte flag, a Boolean value that is `TRUE` if at least one double-byte script system is enabled. (1 byte)

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smKCHRCache`

(A pointer to the cache that stores a copy of the current keyboard-layout (`'KCHR'`) resource 4 bytes).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smRegionCode`

The region code for this localized version of system software, obtained from the system script's international configuration (`'itlc'`) resource. This variable identifies the localized version of the system script. Constants for all defined region codes are listed in "Region Codes A" (page 1798) (2 bytes).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeyDisableState`

The current disable state for keyboards (1 byte). The Script Manager disables some keyboard scripts or keyboard switching when text input must be restricted to certain script systems or when script systems are being moved into or out of the System file. These are the possible values for the variable accessed through `smKeyDisableState`:

- 0All keyboards are enabled; switching is enabled.
- 1Keyboard switching is disabled.
- $FFKeyboards for all non-Roman secondary scripts are disabled

The script management system maintains the keyboard disable state separately for each application. Never set the value of this variable directly with `SetScriptManagerVariable`; call `KeyScript` to change the keyboard disable state for your application.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

Discussion
This section lists and describes the selector constants for accessing the Script Manager variables through calls to the `GetScriptManagerVariable` (page 1724) and `SetScriptManagerVariable` (page 1731) functions. In every case the variable parameter passed to or from the function is a long integer (4 bytes); the number in parentheses indicates how many of the 4 bytes are necessary to hold the input or return value for that variable. If fewer than 4 bytes are needed, the low byte or low word contains the information.

Script Variable Selectors

Specify script variables to get or set using the functions `GetScriptVariable` and `SetScriptVariable`.

```
enum {
    smScriptVersion = 0,
    smScriptMunged = 2,
    smScriptEnabled = 4,
    smScriptRight = 6,
    smScriptJust = 8,
    smScriptRedraw = 10,
    smScriptSysFond = 12,
    smScriptAppFond = 14,
    smScriptBundle = 16,
    smScriptNumber = 16,
    smScriptDate = 18,
    smScriptSort = 20,
    smScriptFlags = 22,
    smScriptToken = 24,
    smScriptEncoding = 26,
    smScriptLang = 28
};
```

Constants

`smScriptVersion`

The script system's version number (2 bytes). When the Script Manager loads the script system, the script system puts its current version number into this variable. The high-order byte contains the major version number, and the low-order byte contains the minor version number.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptMunged`

The modification count for this script system's script variables. (2 bytes)The Script Manager increments the variable accessed by the `smScriptMunged` selector each time the `SetScriptVariable` function is called for this script system. You can check this variable at any time to see whether any of your own data structures that depend on this script system's script variables need to be updated.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptEnabled`

The script-enabled flag, a Boolean value that indicates whether the script has been enabled (1 byte). It is set to $FF when enabled and to 0 when not enabled. Note that this variable is not equivalent to the Script Manager variable accessed by the `smEnabled` selector, which is a count of the total number of enabled script systems.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptRight`

The right-to-left flag, a Boolean value that indicates whether the primary line direction for text in this script is right-to-left or left-to-right (1 byte). It is set to $FF for right-to-left text (used in Arabic and Hebrew script systems) and to 0 for left-to-right (used in Roman and other script systems).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptJust`

The script alignment flag, a byte that specifies the default alignment for text in this script system (1 byte). It is set to $FF for right alignment (common for Arabic and Hebrew), and it is set to 0 for left alignment (common for Roman and other script systems). This flag usually has the same value as the `smScriptRight` flag.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptRedraw`

The script-redraw flag, a byte that provides redrawing recommendations for text of this script system (1 byte). It describes how much of a line should be redrawn when a user adds, inserts, or deletes text. It is set to 0 when only a character should be redrawn (used by the Roman script system), to 1 when an entire word should be redrawn (used by the Japanese script system), and to –1 when the entire line should be redrawn (used by the Arabic and Hebrew script systems). These constants are available for the script-redraw flag:

- `smRedrawChar` (a value of 0)Redraw the character only.
- `smRedrawWord` (a value of 1)Redraw the entire word.
- `smRedrawLine` (a value of –1)Redraw the entire line.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptSysFond`

The preferred system font, the font family ID of the system font preferred for this script (2 bytes). In the Roman script system, this variable specifies Chicago font, whose font family ID is 0 if Roman is the system script. The preferred system font in the Japanese script system is 16384, the font family ID for Osaka.

This variable holds similar information to the variable accessed through the `smScriptSysFondSize` selector. However, changing the value of this variable has no effect on the value accessed through `smScriptSysFondSize`.

Remember that in all localized versions of system software the special value of 0 is remapped to the system font ID. Thus, if an application running under Japanese system software specifies a font family ID of 0 in a function or in the `txFont` field of the current graphics port, Osaka will be used. However, the variable accessed by `smScriptSysFond` will still show the true ID for Osaka (16384).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptAppFond`
: The preferred application font (2 bytes); the font family ID of the application font preferred for this script. In the Roman script system, the value of this variable is the font family ID for Geneva.

  This variable holds similar information to the variable accessed through the `smScriptAppFondSize` selector. However, changing the value of this variable has no effect on the value accessed through `smScriptAppFondSize`.

  Remember that in all localized versions of system software the special value of 1 is remapped to the application font ID. For example, if an application running under Arabic system software specifies a font family ID of 1 in a function, Nadeem will be used. However, the variable accessed by `smScriptSysFond` will still show the true ID for Nadeem (17926).

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptBundle`
: The beginning of `itlb` values.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptNumber`
: The resource ID of the script's numeric-format (`'itl0'`) resource (2 bytes). The numeric-format resource includes formatting information for the correct display of numbers, times, and short dates. The value of this variable is initialized from the script system's international bundle resource.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptDate`
: The resource ID of the script's long-date-format (`'itl1'`) resource (2 bytes). The long-date-format resource includes formatting information for the correct display of long dates (dates that include month or day names). The value of this variable is initialized from the script system's international bundle resource.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptSort`
: The resource ID of the script's string-manipulation (`'itl2'`) resource (2 bytes). The string-manipulation resource contains functions for sorting and tables for word selection, line breaks, character types, and case conversion of text. The value of this variable is initialized from the script system's international bundle resource.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptFlags`
: The script flags word, which contains bit flags specifying attributes of the script (2 bytes). The value of this variable is initialized from the script system's international bundle resource. The "Language Codes A" (page 1782) constants are available for examining attributes in the script flags word. Bits above 8 are nonstatic, meaning that they may change during program execution. (Note that the constant values represent bit numbers in the flags word, not masks.)

  The `smsfIntellCP` flag is set if this script system uses spaces as word delimiters. In such a script system it is possible to implement intelligent cut and paste, in which extra spaces are removed when a word is cut from text, and any needed spaces are added when a word is pasted into text. Macintosh Human Interface Guidelines recommends that you implement intelligent cut and paste in script systems that support it.

  If you use the `CharToPixel` function to determine text widths, such as for line breaking, you need to clear the `smsfReverse` bit first.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptToken`
: The resource ID of the script's tokens (`'itl4'`) resource (2 bytes). The tokens resource contains information for tokenizing and number formatting. The value of this variable is initialized from the script system's international bundle resource.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptEncoding`
: The resource ID of the script's (optional) encoding/rendering (`'itl5'`) resource (2 bytes)For single-byte scripts, the encoding/rendering resource specifies text-rendering behavior for double-byte scripts, it specifies character-encoding information. The value of this variable is taken from the script system's international bundle resource.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptLang`
: The language code for this version of the script. A language is a specialized variation of a specific script system (2 bytes). Constants for all defined language codes are listed in "Language Codes A" (page 1782). The value of this variable is initialized from the script system's international bundle resource.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

Discussion
This section lists and describes the selector constants for accessing script variables through calls to the `GetScriptManagerVariable` (page 1724) and `SetScriptManagerVariable` (page 1731) functions. In every case the variable parameter passed to or from the function is a long integer (4 bytes); the number in parentheses indicates how many of the 4 bytes are necessary to hold the input or return value for that variable. If fewer than 4 bytes are needed, the low byte or low word contains the information.

In many cases the value of a script variable is taken from the script system's international bundle (`'itlb'`) resource.

Script Token Types

Specify script token types.

```
enum {
    tokenIntl = 4,
    tokenEmpty = -1
};
```

Constants

`tokenIntl`

The `'itl'` resource number of the tokenizer.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenEmpty`

Represents an empty flag.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Source Masks

Specify general transliterate text source masks.

```
enum {
    smMaskAll = 0xFFFFFFFF,
    smMaskAscii = 0x00000001,
    smMaskNative = 0x00000002,
    smMaskAscii1 = 0x00000004,
    smMaskAscii2 = 0x00000008,
    smMaskKana1 = 0x00000010,
    smMaskKana2 = 0x00000020,
    smMaskGana2 = 0x00000080,
    smMaskHangul2 = 0x00000100,
    smMaskJamo2 = 0x00000200,
    smMaskBopomofo2 = 0x00000400
};
```

Table Selectors

Specify selectors for the international table

```
enum {
    smWordSelectTable = 0,
    smWordWrapTable = 1,
    smNumberPartsTable = 2,
    smUnTokenTable = 3,
    smWhiteSpaceList = 4,
    iuWordSelectTable = 0,
    iuWordWrapTable = 1,
    iuNumberPartsTable = 2,
    iuUnTokenTable = 3,
    iuWhiteSpaceList = 4
};
```

Constants

`smWordSelectTable`
Specifies to get the word select break table from `'itl2'`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smWordWrapTable`
Specifies to get the word wrap break table from `'itl2'`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smNumberPartsTable`
Specifies to get the default number parts table from `'itl4'`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smUnTokenTable`
Specifies to get the `unToken` table from `'itl4'`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smWhiteSpaceList`
Specifies to get the white space list from `'itl4'`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`iuWordSelectTable`
Obsolete; specifies to get the word select break table from `'itl2'`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`iuWordWrapTable`
Obsolete; specifies to get the word wrap break table from `'itl2'`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`iuNumberPartsTable`
Obsolete; specifies to get the default number parts table from `''itl4'`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`iuUnTokenTable`
Obsolete; specifies to get the `unToken` table from `'itl4'`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`iuWhiteSpaceList`
Obsolete; specifies to get the white space list from `'itl4'`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Discussion
These constants can be used as the value of the `tableCode` variable, passed as a parameter to the `GetIntlResourceTable` (page 1723) function.

Transliteration Target Types 1

Specify transliterate text target types for Roman or for double-byte scripts

```
enum {
    smTransAscii = 0,
    smTransNative = 1,
    smTransCase = 0xFE,
    smTransSystem = 0xFF,
    smTransAscii1 = 2,
    smTransAscii2 = 3,
    smTransKana1 = 4,
    smTransKana2 = 5
};
```

Constants
`smTransAscii`
Specifies to convert to ASCII.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smTransNative`
Specifies to convert to the font script.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smTransCase`
Specifies to convert case for all text.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smTransSystem`
Specifies to convert to the system script.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smTransAscii1`
Specifies to single-byte Roman.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smTransAscii2`
Specifies to double-byte Roman.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smTransKana1`
Specifies to single-byte Japanese Katakana.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smTransKana2`
Specifies to double-byte Japanese Katakana.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

Transliteration Target Types 2

Specify transliteration targets for double-byte script systems.

```
enum {
    smTransGana2 = 7,
    smTransHangul2 = 8,
    smTransJamo2 = 9,
    smTransBopomofo2 = 10,
    smTransLower = 0x4000,
    smTransUpper = 0x8000,
    smTransRuleBaseFormat = 1,
    smTransHangulFormat = 2,
    smTransPreDoubleByting = 1,
    smTransPreLowerCasing = 2
};
```

Constants

`smTransGana2`
Specifies double-byte Japanese Hiragana (no single-byte Hiragana).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smTransHangul2`
Specfies double-byte Korean Hangul.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smTransJamo2`
Specifies double-byte Korean Jamo.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smTransBopomofo2`
Specifies double-byte Chinese Bopomofo.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smTransLower`
Specifies target becomes lowercase.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smTransUpper`
Specifies target becomes uppercase .
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smTransRuleBaseFormat`
Specifies rule-based `trsl` resource format.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smTransHangulFormat`
Specifies table-based Hangul `trsl` resource format.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smTransPreDoubleByting`
Specifies to convert all text to double byte before transliteration.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smTransPreLowerCasing`
Specifies to convert all text to lower case before transliteration.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

Language Codes

Language Codes A

Specify language codes (values 0 though 23).

```
enum {
    langEnglish = 0,
    langFrench = 1,
    langGerman = 2,
    langItalian = 3,
    langDutch = 4,
    langSwedish = 5,
    langSpanish = 6,
    langDanish = 7,
    langPortuguese = 8,
    langNorwegian = 9,
    langHebrew = 10,
    langJapanese = 11,
    langArabic = 12,
    langFinnish = 13,
    langGreek = 14,
    langIcelandic = 15,
    langMaltese = 16,
    langTurkish = 17,
    langCroatian = 18,
    langTradChinese = 19,
    langUrdu = 20,
    langHindi = 21,
    langThai = 22,
    langKorean = 23
};
```

Constants

`langEnglish`
Represents the English language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langFrench`
Represents the French language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langGerman`
Represents the German language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langItalian`
Represents the Italian language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langDutch`
Represents the Dutch language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langSwedish`
Represents the Swedish language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langSpanish`
Represents the Spanish language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langDanish`
Represents the Danish language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langPortuguese`
Represents the Portuguese language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langNorwegian`
Represents the Norwegian language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langHebrew`
Represents the Hebrew language. The associated script code is `smHebrew`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langJapanese`
Represents the Japanese language. The associated script code is `smJapanese`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langArabic`
Represents the Arabic language. The associated script code is `smArabic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langFinnish`
Represents the Finnish language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langGreek`
Represents the Greek language. The associated script code is `smGreek`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langIcelandic`
Represents the Icelandic language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langMaltese`
Represents the Maltese language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langTurkish`
Represents the Turkish language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langCroatian`
Represents the Croatian language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langTradChinese`
Represents the Chinese (traditional chararacters) language. The associated script code is `smTradChinese`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langUrdu`
Represents the Urdu language. The associated script code is `smArabic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langHindi`
Represents the Hindi language. The associated script code is `smDevanagari`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langThai`
Represents the Thai language. The associated script code is `smThai`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langKorean`
Represents the Korean language. The associated script code is `smKorean`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

Language Codes B

Specify language codes (values 24 though 46).

```
enum {
    langLithuanian = 24,
    langPolish = 25,
    langHungarian = 26,
    langEstonian = 27,
    langLatvian = 28,
    langSami = 29,
    langFaroese = 30,
    langFarsi = 31,
    langPersian = 31,
    langRussian = 32,
    langSimpChinese = 33,
    langFlemish = 34,
    langIrishGaelic = 35,
    langAlbanian = 36,
    langRomanian = 37,
    langCzech = 38,
    langSlovak = 39,
    langSlovenian = 40,
    langYiddish = 41,
    langSerbian = 42,
    langMacedonian = 43,
    langBulgarian = 44,
    langUkrainian = 45,
    langByelorussian = 46,
    langBelorussian = 46
};
```

Constants

`langLithuanian`
Represents the Lithuanian language. The associated script code is `smEastEurRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langPolish`
Represents the Polish language. The associated script code is `smEastEurRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langHungarian`
Represents the Hungarian language. The associated script code is `smEastEurRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langEstonian`
Represents the Estonian language. The associated script code is `smEastEurRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langLatvian`
Represents the Lettish language. The associated script code is `smEastEurRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langSami`
Represents the language of the Sami people of northern Scandinavia.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langFaroese`
Modified `smRoman`/Icelandic script
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langFarsi`
Represents the Farsi language. The associated script code is `smArabic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langPersian`
Represents the Farsi language. The associated script code is `smArabic`. This is the same as the language code `langFarsi`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langRussian`
Represents the Russian language. The associated script code is `smCyrillic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langSimpChinese`
Represents the Chinese (simplified chararacters) language. The associated script code is `smSimpChinese`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langFlemish`
Represents the Flemish language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langIrishGaelic`
Represents Irish Gaelic. The associated script code is `smRoman` or modified `smRoman`/Celtic script (without dot above).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langAlbanian`
Represents the Albanian language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langRomanian`
Represents the Romanian language. The associated script code is `smEastEurRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langCzech`
Represents the Czech language. The associated script code is `smEastEurRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langSlovak`
Represents the Slovak language. The associated script code is `smEastEurRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langSlovenian`
Represents the Slovenian language. The associated script code is `smEastEurRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langYiddish`
Represents the Yiddish language. The associated script code is `smHebrew`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langSerbian`
Represents the Serbian language. The associated script code is `smCyrillic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langMacedonian`
Represents the Macedonian language. The associated script code is `smCyrillic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langBulgarian`
Represents the Bulgarian language. The associated script code is `smCyrillic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langUkrainian`
Represents the Ukrainian language. The associated script code is `smCyrillic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langByelorussian`
Represents the Byelorussian language. The associated script code is `smCyrillic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langBelorussian`
Represents a synonym for `langByelorussian`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

Language Codes C

Specify language codes (values 47 though 70).

```
enum {
    langUzbek = 47,
    langKazakh = 48,
    langAzerbaijani = 49,
    langAzerbaijanAr = 50,
    langArmenian = 51,
    langGeorgian = 52,
    langMoldavian = 53,
    langKirghiz = 54,
    langTajiki = 55,
    langTurkmen = 56,
    langMongolian = 57,
    langMongolianCyr = 58,
    langPashto = 59,
    langKurdish = 60,
    langKashmiri = 61,
    langSindhi = 62,
    langTibetan = 63,
    langNepali = 64,
    langSanskrit = 65,
    langMarathi = 66,
    langBengali = 67,
    langAssamese = 68,
    langGujarati = 69,
    langPunjabi = 70
};
```

Constants

`langUzbek`

Represents the Uzbek language. The associated script code is `smCyrillic`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langKazakh`

Represents the Kazakh language. The associated script code is `smCyrillic`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langAzerbaijani`

Represents the Azerbaijani language. The associated script code is `smCyrillic`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langAzerbaijanAr`

Represents the Azerbaijani language. The associated script code is `smArabic`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langArmenian`
Represents the Armenian language. The associated script code is `smArmenian`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langGeorgian`
Represents the Georgian language. The associated script code is `smGeorgian`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langMoldavian`
Represents the Moldovan language. The associated script code is `smCyrillic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langKirghiz`
Represents the Kirghiz language. The associated script code is `smCyrillic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langTajiki`
Represents the Tajiki language. The associated script code is `smCyrillic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langTurkmen`
Represents the Turkmen language. The associated script code is `smCyrillic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langMongolian`
Represents the Mongolian language. The associated script code is `smMongolian`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langMongolianCyr`
Represents the Mongolian language. The associated script code is `smCyrillic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langPashto`
Represents the Pashto language. The associated script code is `smArabic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langKurdish`
Represents the Kurdish language. The associated script code is `smArabic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langKashmiri`
Represents the Kashmiri language. The associated script code is `smArabic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langSindhi`
Represents the Sindhi language. The associated script code is `smExtArabic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langTibetan`
Represents the Tibetan language. The associated script code is `smTibetan`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langNepali`
Represents the Nepali language. The associated script code is `smDevanagari`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langSanskrit`
Represents the Sanskrit language. The associated script code is `smDevanagari`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langMarathi`
Represents the Marathi language. The associated script code is `smDevanagari`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langBengali`
Represents the Bengali language. The associated script code is `smBengali`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langAssamese`
Represents the Assamese language. The associated script code is `smBengali`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langGujarati`
Represents the Gujarati language. The associated script code is `smGujarati`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langPunjabi`
Represents the Punjabi language. The associated script code is `smGurmukhi`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

Language Codes D

Specify language codes (values 71 though 94).

```
enum {
    langOriya = 71,
    langMalayalam = 72,
    langKannada = 73,
    langTamil = 74,
    langTelugu = 75,
    langSinhalese = 76,
    langBurmese = 77,
    langKhmer = 78,
    langLao = 79,
    langVietnamese = 80,
    langIndonesian = 81,
    langTagalog = 82,
    langMalayRoman = 83,
    langMalayArabic = 84,
    langAmharic = 85,
    langTigrinya = 86,
    langOromo = 87,
    langSomali = 88,
    langSwahili = 89,
    langKinyarwanda = 90,
    langRuanda = 90,
    langRundi = 91,
    langNyanja = 92,
    langChewa = 92,
    langMalagasy = 93,
    langEsperanto = 94
};
```

Constants

`langOriya`
Represents the Oriya language. The associated script code is `smOriya`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langMalayalam`
Represents the Malayalam language. The associated script code is `smMalayalam`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langKannada`
Represents the Kannada language. The associated script code is `smKannada`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langTamil`
Represents the Tamil language. The associated script code is `smTamil`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langTelugu`
Represents the Telugu language. The associated script code is `smTelugu`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langSinhalese`
Represents the Sinhalese language. The associated script code is `smSinhalese`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langBurmese`
Represents the Burmese language. The associated script code is `smBurmese`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langKhmer`
Represents the Khmer language. The associated script code is `smKhmer`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langLao`
Represents the Lao language. The associated script code is `smLaotian`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langVietnamese`
Represents the Vietnamese language. The associated script code is `smVietnamese`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langIndonesian`
Represents the Indonesian language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langTagalog`
Represents the Tagalog language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langMalayRoman`
Represents the Malay language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langMalayArabic`
Represents the Malay language. The associated script code is `smArabic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langAmharic`
Represents the Amharic language. The associated script code is `smEthiopic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langTigrinya`
Represents the Tigrinya language. The associated script code is `smEthiopic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langOromo`
Represents the Galla language. The associated script code is `smEthiopic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langSomali`
Represents the Somali language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langSwahili`
Represents the Swahili language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langKinyarwanda`
The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langRuanda`
Represents the Ruanda language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langRundi`
Represents the Rundi language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langNyanja`
The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langChewa`
Represents the Chewa language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langMalagasy`
Represents the Malagasy language. The associated script code is `smRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langEsperanto`
Represents the Esperanto language. The associated script code is `smRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Language Codes E

Specify lanaguage codes (values 128 though 141).

```
enum {
    langWelsh = 128,
    langBasque = 129,
    langCatalan = 130,
    langLatin = 131,
    langQuechua = 132,
    langGuarani = 133,
    langAymara = 134,
    langTatar = 135,
    langUighur = 136,
    langDzongkha = 137,
    langJavaneseRom = 138,
    langSundaneseRom = 139,
    langGalician = 140,
    langAfrikaans = 141
};
```

Constants

`langWelsh`
Represents the Welsh language. The associated script code is `smRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langBasque`
Represents the Basque language. The associated script code is `smRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langCatalan`
Represents the Catalan language. The associated script code is `smRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langLatin`
Represents the Latin language. The associated script code is `smRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langQuechua`
Represents the Quechua language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langGuarani`
Represents the Guarani language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langAymara`
Represents the Aymara language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langTatar`
Represents the Tatar language. The associated script code is `smCyrillic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langUighur`
Represents the Uighar language. The associated script code is `smArabic`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langDzongkha`
Represents the Bhutanese language. The associated script code is `smTibetan`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langJavaneseRom`
Represents the Javanese language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langSundaneseRom`
Represents the Sundanese language. The associated script code is `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langGalician`
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langAfrikaans`
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

Language Codes F

Specify language codes (values 142 through 150).

```
enum {
    langBreton = 142,
    langInuktitut = 143,
    langScottishGaelic = 144,
    langManxGaelic = 145,
    langIrishGaelicScript = 146,
    langTongan = 147,
    langGreekPoly = 148,
    langGreenlandic = 149,
    langAzerbaijanRoman = 150
};
```

Constants

`langBreton`
The associated script code is `smRoman` or modified `smRoman`/Celtic script
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langInuktitut`
Inuit script using `smEthiopic` script code
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langScottishGaelic`
The associated script code is smRoman or modified `smRoman`/Celtic script
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langManxGaelic`
The associated script code is `smRoman` or modified `smRoman`/Celtic script
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langIrishGaelicScript`
The associated script code is modified `smRoman`/Gaelic script (using dot above).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langTongan`
The associated script code is `smRoman` script
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langGreekPoly`
The associated script code is `smGreek` script
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langGreenlandic`
The associated script code is `smRoman` script
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langAzerbaijanRoman`
Represents the Azerbaijani language. The associated script code is `Roman` script.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Language Code - Unspecified

Indicates the language is not specified.

```
enum {
    langUnspecified = 32767
};
```

Region Codes

Range Checking Region Code

Specify values for the the minimum and maximum defined region codes.

```
enum {
    minCountry = verUS,
    maxCountry = verGreenland
};
```

Constants

`minCountry`
The lowest defined region code (for range-checking); currently this is equal to the region code `verUS`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`maxCountry`
The highest defined region code (for range-checking); currently this is equal to the region code `verThailand`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Region Codes A

Specify codes for a variety of regions (values 0 - 25).

```
enum {
    verUS = 0,
    verFrance = 1,
    verBritain = 2,
    verGermany = 3,
    verItaly = 4,
    verNetherlands = 5,
    verFlemish = 6,
    verSweden = 7,
    verSpain = 8,
    verDenmark = 9,
    verPortugal = 10,
    verFrCanada = 11,
    verNorway = 12,
    verIsrael = 13,
    verJapan = 14,
    verAustralia = 15,
    verArabic = 16,
    verFinland = 17,
    verFrSwiss = 18,
    verGrSwiss = 19,
    verGreece = 20,
    verIceland = 21,
    verMalta = 22,
    verCyprus = 23,
    verTurkey = 24,
    verYugoCroatian = 25
};
```

Constants

`verUS`
Represents the region of the United States.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verFrance`
Represents the region of France.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verBritain`
Represents the region of Great Britain.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verGermany`
Represents the region of Germany.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verItaly`
Represents the region of Italy.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verNetherlands`
Represents the region of the Netherlands.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verFlemish`
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verSweden`
Represents the region of Sweden.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verSpain`
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verDenmark`
Represents the region of Denmark.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verPortugal`
Represents the region of Portugal.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verFrCanada`
Represents the French Canadian region.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verNorway`
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verIsrael`
Represents the region of Israel.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verJapan`
Represents the region of Japan.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verAustralia`
Represents the region of Australia.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verArabic`
Represents the Arabic world. This is the same as the region code `verArabia`.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verFinland`
Represents the region of Finland.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verFrSwiss`
Represents French for the region of Switzerland.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verGrSwiss`
Represents German for the region of Switzerland.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verGreece`
Represents the region of Greece.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verIceland`
Represents the region of Iceland.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verMalta`
Represents the region of Malta.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verCyprus`
Represents the region of Cyprus.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verTurkey`
Represents the region of Turkey.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verYugoCroatian`
Represents the Croatian system for the region of Yugoslavia.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

Discussion
Each region is associated with a particular language code and script code (not shown). The existence of a defined region code does not necessarily imply the existence of a version of Macintosh system software localized for that region.

Region Codes B

Specify region codes (values 26 though 32).

```
enum {
    verNetherlandsComma = 26,
    verBelgiumLuxPoint = 27,
    verCanadaComma = 28,
    verCanadaPoint = 29,
    vervariantPortugal = 30,
    vervariantNorway = 31,
    vervariantDenmark = 32
};
```

Constants

`verNetherlandsComma`
Specifies Dutch.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verBelgiumLuxPoint`
Specifies Belgium.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verCanadaComma`
Specifies Canadian ISO.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verCanadaPoint`
Specifies Canadian; now unused.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`vervariantPortugal`
Unused.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`vervariantNorway`
Unused.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`vervariantDenmark`
Specifies Danish Mac Plus.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Region Codes C

Specify region codes (values 33 through 61).

```
enum {
    verIndiaHindi = 33,
    verPakistanUrdu = 34,
    verTurkishModified = 35,
    verItalianSwiss = 36,
    verInternational = 37,
    verRomania = 39,
    verGreecePoly = 40,
    verLithuania = 41,
    verPoland = 42,
    verHungary = 43,
    verEstonia = 44,
    verLatvia = 45,
    verSami = 46,
    verFaroeIsl = 47,
    verIran = 48,
    verRussia = 49,
    verIreland = 50,
    verKorea = 51,
    verChina = 52,
    verTaiwan = 53,
    verThailand = 54,
    verScriptGeneric = 55,
    verCzech = 56,
    verSlovak = 57,
    verFarEastGeneric = 58,
    verMagyar = 59,
    verBengali = 60,
    verByeloRussian = 61
};
```

Constants

`verIndiaHindi`
The Hindi system for the region of India; hi_IN..

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`verPakistanUrdu`
Urdu for Pakistan; ur_PK.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`verTurkishModified`
Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`verItalianSwiss`
Italian Swiss; it_CH.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verInternational`
English for international use; Z en.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verRomania`
Romaniza; ro_RO
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verGreecePoly`
Polytonic Greek (classical); grc.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verLithuania`
Lithuania; lt_LT.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verPoland`
Poland; pl_PL.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verHungary`
Represents the region of Hungary; hu_HU.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verEstonia`
Represents the region of Estonia; et_EE.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verLatvia`
Represents the region of Latvia; lv_LV.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verSami`
se.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verFaroeIsl`
fo_FO.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verIran`
Persian/Farsi Represents the region of Iran; fa_IR .
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verRussia`
Represents the region of Russia; ru_RU.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verIreland`
Represents Irish Gaelic for Ireland (without dot above); ga_IE.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verKorea`
Represents the region of Korea; ko_KR.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verChina`
Simplified Chinese; zh_CN.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verTaiwan`
Traditional Chinese; zh_TW.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verThailand`
Represents the region of Thailand; th_TH.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verScriptGeneric`
Generic script system (no language or script).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verCzech`
cs_CZ.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verSlovak`
sk_SK.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`verFarEastGeneric`
Generic Far East system (no language or script).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`verMagyar`
Unused; see `verHungary`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`verBengali`
Bangladesh or India; bn.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`verByeloRussian`
be_B,

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Region Codes D

Specify region codes (values 62 through 97).

```
enum {
    verUkraine = 62,
    verGreeceAlt = 64,
    verSerbian = 65,
    verSlovenian = 66,
    verMacedonian = 67,
    verCroatia = 68,
    verGermanReformed = 70,
    verBrazil = 71,
    verBulgaria = 72,
    verCatalonia = 73,
    verMultilingual = 74,
    verScottishGaelic = 75,
    verManxGaelic = 76,
    verBreton = 77,
    verNunavut = 78,
    verWelsh = 79,
    verIrishGaelicScript = 81,
    verEngCanada = 82,
    verBhutan = 83,
    verArmenian = 84,
    verGeorgian = 85,
    verSpLatinAmerica = 86,
    verTonga = 88,
    verFrenchUniversal = 91,
    verAustria = 92,
    verGujarati = 94,
    verPunjabi = 95,
    verIndiaUrdu = 96,
    verVietnam = 97
};
```

Regions Codes E

Specify region codes (values 98 through 109).

```
enum {
    verFrBelgium = 98,
    verUzbek = 99,
    verSingapore = 100,
    verNynorsk = 101,
    verAfrikaans = 102,
    verEsperanto = 103,
    verMarathi = 104,
    verTibetan = 105,
    verNepal = 106,
    verGreenland = 107,
    verIrelandEnglish = 108
};
```

Token Constants

Tokens - Mathematical

Specify tokens used in mathematical operations.

```
enum {
    tokenLeftCurly = 20,
    tokenRightCurly = 21,
    tokenLeftEnclose = 22,
    tokenRightEnclose = 23,
    tokenPlus = 24,
    tokenMinus = 25,
    tokenAsterisk = 26,
    tokenDivide = 27,
    tokenPlusMinus = 28,
    tokenSlash = 29,
    tokenBackSlash = 30,
    tokenLess = 31,
    tokenGreat = 32,
    tokenEqual = 33,
    tokenLessEqual2 = 34,
    tokenLessEqual1 = 35,
    tokenGreatEqual2 = 36,
    tokenGreatEqual1 = 37,
    token2Equal = 38,
    tokenColonEqual = 39
};
```

Constants

`tokenLeftCurly`

Represents an opening curly bracket.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenRightCurly`

Represents a closing curly bracket.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenLeftEnclose`
Represents an opening European double quote.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenRightEnclose`
Represents a closing European double quote.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenPlus`
Represents a plus sign.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenMinus`
Represents a minus sign.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenAsterisk`
Represents a times/multiply sign.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenDivide`
Represents a divide.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenPlusMinus`
Represents a plus-or-minus symbol.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenSlash`
Represents a slash.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenBackSlash`
Represents a backslash.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenLess`
Represents a less than sign.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenGreat`
Represents a greater than sign.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenEqual`
Represents an equal.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenLessEqual2`
Represents a less than or equal to sign (2 symbols).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenLessEqual1`
Represents a less than or equal to sign (1 symbol).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenGreatEqual2`
Represents a greater than or equal to sign (2 symbols).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenGreatEqual1`
Represents a greater-than-or-equal-to sign (1 symbol).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`token2Equal`
Represents a double equal sign.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenColonEqual`
Represents a colon equal sign.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

Tokens - Punctuation

Specify tokens for various punctuation marks.

```
enum {
    tokenNotEqual = 40,
    tokenLessGreat = 41,
    tokenExclamEqual = 42,
    tokenExclam = 43,
    tokenTilde = 44,
    tokenComma = 45,
    tokenPeriod = 46,
    tokenLeft2Quote = 47,
    tokenRight2Quote = 48,
    tokenLeft1Quote = 49,
    tokenRight1Quote = 50,
    token2Quote = 51,
    token1Quote = 52,
    tokenSemicolon = 53,
    tokenPercent = 54,
    tokenCaret = 55,
    tokenUnderline = 56,
    tokenAmpersand = 57,
    tokenAtSign = 58,
    tokenBar = 59
};
```

Constants

`tokenNotEqual`
Represents a not equal sign.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenLessGreat`
Represents a less/greater sign (not equal in Pascal).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenExclamEqual`
Represents an exclamation equal sign (not equal in C).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenExclam`
Represents as exclamation point.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenTilde`
Represents a centered tilde.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenComma`
Represents a comma.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenPeriod`
Represents a period.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenLeft2Quote`
Represents an opening double quote.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenRight2Quote`
Represents a closing double quote.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenLeft1Quote`
Represents an opening single quote.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenRight1Quote`
Represents a closing single quote.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`token2Quote`
Represents a double quote.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`token1Quote`
Represents a single quote.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenSemicolon`
Represents a semicolon.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenPercent`
Represents a percent sign.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenCaret`
Represents a caret.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenUnderline`
Represents an underline.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenAmpersand`
Represents an ampersand.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenAtSign`
Represents an at sign.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenBar`
Represents a vertical bar.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

Tokens for Symbols

Specify tokens for various symbols.

```
enum {
    tokenQuestion = 60,
    tokenPi = 61,
    tokenRoot = 62,
    tokenSigma = 63,
    tokenIntegral = 64,
    tokenMicro = 65,
    tokenCapPi = 66,
    tokenInfinity = 67,
    tokenColon = 68,
    tokenHash = 69,
    tokenDollar = 70,
    tokenNoBreakSpace = 71,
    tokenFraction = 72,
    tokenIntlCurrency = 73,
    tokenLeftSingGuillemet = 74,
    tokenRightSingGuillemet = 75,
    tokenPerThousand = 76,
    tokenEllipsis = 77,
    tokenCenterDot = 78,
    tokenNil = 127
};
```

Constants

`tokenQuestion`
Represents a question mark.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenPi`
Represents a Pi token.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenRoot`
Represents a square root sign.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenSigma`
Represents a capital sigma.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenIntegral`
Represents an integral sign.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenMicro`
Represents a micro.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenCapPi`
Represents a capital pi.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenInfinity`
Represents an infinity sign.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenColon`
Represents a colon.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenHash`
Represents a pound sign (U.S. weight).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenDollar`
Represents a dollar sign.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenNoBreakSpace`
Represents a nonbreaking space.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenFraction`
Represents a fraction.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenIntlCurrency`
Represents an international currency token.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenLeftSingGuillemet`
Represents an opening single guillemet.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenRightSingGuillemet`
Represents a closing single guillemet.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenPerThousand`
Represents a per thousands token.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenEllipsis`
Represents an ellipsis character.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenCenterDot`
Represents a center dot.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenNil`
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

Token Types

Specify types of tokens.

```
enum {
    tokenUnknown = 0,
    tokenWhite = 1,
    tokenLeftLit = 2,
    tokenRightLit = 3,
    tokenAlpha = 4,
    tokenNumeric = 5,
    tokenNewLine = 6,
    tokenLeftComment = 7,
    tokenRightComment = 8,
    tokenLiteral = 9,
    tokenEscape = 10,
    tokenAltNum = 11,
    tokenRealNum = 12,
    tokenAltReal = 13,
    tokenReserve1 = 14,
    tokenReserve2 = 15,
    tokenLeftParen = 16,
    tokenRightParen = 17,
    tokenLeftBracket = 18,
    tokenRightBracket = 19
};
```

Constants

`tokenUnknown`
Has no existing token type.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenWhite`
Represents a whitespace character.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenLeftLit`
Represents an opening literal marker.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenRightLit`
Represents a closing literal marker.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenAlpha`
Represents an alphabetic token.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenNumeric`
Represents a numeric token.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenNewLine`
Represents a new line.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenLeftComment`
Represents an opening comment marker.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenRightComment`
Represents a closing comment marker.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenLiteral`
Represents a literal token.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenEscape`
Represents an escape character.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenAltNum`
Represents an alternate number (such as at $B0–$B9).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenRealNum`
Represents a real number.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenAltReal`
Represents an alternate real number.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenReserve1`
Reserved.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenReserve2`
Reserved.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenLeftParen`
Represents an opening parenthesis.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenRightParen`
Represents a closing parenthesis.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenLeftBracket`
Represents an opening square bracket.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenRightBracket`
Represents a closing square bracket.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

Token Results

Specify token conditions returned by the function `IntlTokenize`.

```
enum {
    tokenOK = 0,
    tokenOverflow = 1,
    stringOverflow = 2,
    badDelim = 3,
    badEnding = 4,
    crash = 5
};
typedef SInt8 TokenResults;
```

Constants

`tokenOK`
Indicates the function exectured without error.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`tokenOverflow`
Indicates a token overflow.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`stringOverflow`
Indicates a string overflow.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`badDelim`
Indicates a bad delimiter,

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`badEnding`
Indicates a bad ending.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`crash`
Indicates a crash.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Discussion
Token results are returned by the function `IntlTokenize` (page 1729).

Obsolete Constants

Obsolete Language Codes

Specify obsolete language codes provided for backward compatibility.

```
enum {
    langPortugese = 8,
    langMalta = 16,
    langYugoslavian = 18,
    langChinese = 19,
    langLettish = 28,
    langLapponian = 29,
    langLappish = 29,
    langSaamisk = 29,
    langFaeroese = 30,
    langIrish = 35,
    langGalla = 87,
    langAfricaans = 141
};
```

Discussion
These are obsolete language code names kept for backward compatibility. They have one or more of the following problems: misspelled, ambiguous, misleading, archaic, inappropriate.

Obsolete Regions Codes

Specfiy obsolete region code names provided for backward compatibility.

```
enum {
    verFrBelgiumLux = 6,
    verBelgiumLux = 6,
    verArabia = 16,
    verYugoslavia = 25,
    verIndia = 33,
    verPakistan = 34,
    verRumania = 39,
    verGreekAncient = 40,
    verLapland = 46,
    verFaeroeIsl = 47,
    verGenericFE = 58,
    verBelarus = 61,
    verUkrania = 62,
    verAlternateGr = 64,
    verSerbia = 65,
    verSlovenia = 66,
    verMacedonia = 67,
    verBrittany = 77,
    verWales = 79,
    verArmenia = 84,
    verGeorgia = 85,
    verAustriaGerman = 92,
    verTibet = 105
};
```

Discussion
Obsolete region code names (kept for backward compatibility): Misspelled or alternate form, ambiguous, misleading, considered pejorative, archaic, etc.

Obsolete Roman Script Constants

Specify obsolete constants provided for backward compatibility.

```
enum {
    romanSysFond = 0x3FFF,
    romanAppFond = 3,
    romanFlags = 0x0007,
    smFondStart = 0x4000,
    smFondEnd = 0xC000,
    smUprHalfCharSet = 0x80
};
```

Discussion
You should use the function `GetScriptVariable` (page 1726) to obtain the information specified by these constants.

Obsolete Script Codes

Specify obsolete script code names provided for backward compatibility.

```
enum {
    smChinese = 2,
    smRussian = 7,
    smLaotian = 22,
    smAmharic = 28,
    smSlavic = 29,
    smEastEurRoman = 29,
    smSindhi = 31,
    smKlingon = 32
};
```

Obsolete System Script Codes

Specify obsolete script code values for International Utilities provided for backward compatibility.

```
enum {
    iuSystemScript = -1,
    iuCurrentScript = -2
};
```

Obsolete Token Codes

Specify obsolete token names provided for backward compatibility.

```
enum {
    delimPad = -2,
    tokenTilda = 44,
    tokenCarat = 55
};
```

Result Codes

The most common result codes returned by Script Manager are listed below.

Result Code	Value	Description
`smNotInstalled`	0	Routine not available in script. Available in Mac OS X v10.0 and later.
`smBadVerb`	-1	Bad verb passed to a routine. Available in Mac OS X v10.0 and later.
`smBadScript`	-2	Bad script code passed to a routine. Available in Mac OS X v10.0 and later.

SCSI Manager Reference (Not Recommended)

Framework: CoreServices/CoreServices.h

Declared in SCSI.h

Overview

Carbon supports only the `SCSIAction` function in the SCSI Manager, although this function is no longer recommended. For Mac OS X, the I/O Kit should be used to support more complex SCSI devices.

Important: The SCSI Manager is deprecated in Mac OS X v10.2 and later. You should use I/O Kit to support SCSI devices instead. For more information, see *SCSI Architecture Model Device Interface Guide*.

Functions

DisposeSCSICallbackUPP

Disposes of a UPP to a completion routine. (Deprecated in Mac OS X v10.2. There is no replacement function. For details about communicating with SCSI devices in Mac OS X v10.2 and later, see *SCSI Architecture Model Device Interface Guide*.)

Not recommended

```
void DisposeSCSICallbackUPP (
   SCSICallbackUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.2.
Not available to 64-bit applications.

Declared In
`SCSI.h`

InvokeSCSICallbackUPP

Calls your completion routine. (Deprecated in Mac OS X v10.2. There is no replacement function. For details about communicating with SCSI devices in Mac OS X v10.2 and later, see *SCSI Architecture Model Device Interface Guide*.)

Not recommended

```
void InvokeSCSICallbackUPP (
   void *scsiPB,
   SCSICallbackUPP userUPP
);
```

Discussion
You should not have to call the `InvokeSCSICallbackUPP` function as the system calls your completion routine for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.2.
Not available to 64-bit applications.

Declared In
`SCSI.h`

NewSCSICallbackUPP

Creates a new universal procedure pointer (UPP) to a completion routine. (Deprecated in Mac OS X v10.2. There is no replacement function. For details about communicating with SCSI devices in Mac OS X v10.2 and later, see *SCSI Architecture Model Device Interface Guide*.)

Not recommended

```
SCSICallbackUPP NewSCSICallbackUPP (
   SCSICallbackProcPtr userRoutine
);
```

Return Value
See the description of the `SCSICallbackUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.2.
Not available to 64-bit applications.

Declared In
`SCSI.h`

SCSIAction

Initiates a SCSI transaction or request a service from the XPT or SIM. (Deprecated in Mac OS X v10.2. There is no replacement function. For details about communicating with SCSI devices in Mac OS X v10.2 and later, see *SCSI Architecture Model Device Interface Guide*.)

Not recommended

```
OSErr SCSIAction (
   SCSI_PB *parameterBlock
);
```

Parameters

parameterBlock

A pointer to a SCSI Manager parameter block.

Return Value

A result code. See "SCSI Manager Result Codes" (page 1863).

Discussion

The `SCSIAction` function initiates the request specified by the `scsiFunctionCode` field of the parameter block. Certain types of requests are handled by the XPT, but most are handled by the SIM.

When called asynchronously, `SCSIAction` normally returns the `NoErr` result code, indicating that the request was queued successfully. The result of the SCSI transaction is returned in the `scsiResult` field upon completion. If the `SCSIAction` function returns an error code, the request was not queued and the completion routine will not be called.

When the completion routine is called, it receives the A5 world that existed when the `SCSIAction` request was received. If A5 was invalid when the request was made, it is also invalid in the completion routine.

Your completion routine should use the following function prototype:

```
pascal void (*CallbackProc) (void * scsiPB);
```

There is no implied ordering of asynchronous requests made to different devices. An earlier request may be started later, and a later request may complete earlier. However, a series of requests to the same device is issued to that device in the order received, except when the `scsiSIMQHead` flag is set in the `scsiFlags` field of the parameter block.

When called synchronously, the `SCSIAction` function returns the actual result of the operation. It also places this result in the `scsiResult` field.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.2.

Not available to 64-bit applications.

Declared In

`SCSI.h`

Callbacks

SCSICallbackProcPtr

Defines a pointer to a completion routine.

```
typedef void (*SCSICallbackProcPtr) (
    void * scsiPB
);
```

If you name your function `MySCSICallbackProc`, you would declare it like this:

```
void MySCSICallbackProc (
    void * scsiPB
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`SCSI.h`

Data Types

CDB

You use the command descriptor block record to pass SCSI commands to the `SCSIAction` function.

```
union CDB {
    BytePtr cdbPtr;
    UInt8 cdbBytes[16];
};
typedef union CDB CDB;
typedef CDB * CDBPtr;
```

Fields

`cdbPtr`
A pointer to a buffer containing a CDB.

`cdbBytes`
A buffer in which you can place a CDB.

Discussion
The SCSI commands can be stored within this structure, or you can provide a pointer to them. You set the `scsiCDBIsPointer` flag in the SCSI parameter block if this record contains a pointer.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`SCSI.h`

DeviceIdent

You use the device identification record to specify a target device by its bus, SCSI ID, and logical unit number (LUN).

```
struct DeviceIdent {
    UInt8 diReserved;
    UInt8 bus;
    UInt8 targetID;
    UInt8 LUN;
};
typedef struct DeviceIdent DeviceIdent;
```

Fields

`diReserved`
Reserved.

`bus`
The bus number of the SIM/HBA for the target device.

`targetID`
The SCSI ID number of the target device.

`LUN`
The target LUN, or 0 if the device does not support logical units.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`SCSI.h`

DeviceIdentATA

```
struct DeviceIdentATA {
    UInt8 diReserved;
    UInt8 busNum;
    UInt8 devNum;
    UInt8 diReserved2;
};
typedef struct DeviceIdentATA DeviceIdentATA;
```

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
`SCSI.h`

SCSI_PB

You use the SCSI Manager parameter block to pass information to the `SCSIAction` function.

```
struct SCSI_PB {
    SCSIHdr * qLink;
    short scsiReserved1;
    UInt16 scsiPBLength;
    UInt8 scsiFunctionCode;
    UInt8 scsiReserved2;
    volatile OSErr scsiResult;
    DeviceIdent scsiDevice;
    SCSICallbackUPP scsiCompletion;
    UInt32 scsiFlags;
    BytePtr scsiDriverStorage;
    Ptr scsiXPTprivate;
    long scsiReserved3;
};
typedef struct SCSI_PB SCSI_PB;
```

Fields

`qLink`
: A pointer to the next entry in the request queue. This field is used internally by the SCSI Manager and must be set to 0 when the parameter block is initialized. The SCSI Manager functions always set this field to 0 before returning, so you do not need to set it to 0 again before reusing a parameter block.

`scsiReserved1`
: Reserved.

`scsiPBLength`
: The size of the parameter block, in bytes, including the parameter block header.

`scsiFunctionCode`
: A function selector code that specifies the service being requested.

`scsiReserved2`
: Reserved.

`scsiResult`
: The result code returned by the XPT or SIM when the function completes. The value `scsiRequestInProgress` indicates that the request is still in progress or queued.

`scsiDevice`
: A 4-byte value that uniquely identifies the target device for a request. The `DeviceIdent` data type designates the bus number, target SCSI ID, and logical unit number (LUN).

`scsiCompletion`
: A pointer to a completion routine.

`scsiFlags`
: Flags indicating the transfer direction and any special handling required for this request.

`scsiDriverStorage`
: A pointer to the device driver's private storage. This field is not affected or used by the SCSI Manager.

`scsiXPTprivate`
: Private field for use in XPT.

`scsiReserved3`
: Reserved.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
SCSI.h

SCSICallbackUPP

Defines a universal procedure pointer (UPP) to a completion routine.

```
typedef SCSICallbackProcPtr SCSICallbackUPP;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
SCSI.h

SCSI_IO

Defines the SCSI I/O parameter block.

```
struct SCSI_IO {
    SCSIHdr * qLink;
    short scsiReserved1;
    UInt16 scsiPBLength;
    UInt8 scsiFunctionCode;
    UInt8 scsiReserved2;
    volatile OSErr scsiResult;
    DeviceIdent scsiDevice;
    SCSICallbackUPP scsiCompletion;
    UInt32 scsiFlags;
    BytePtr scsiDriverStorage;
    Ptr scsiXPTprivate;
    long scsiReserved3;
    UInt16 scsiResultFlags;
    UInt16 scsiReserved3pt5;
    BytePtr scsiDataPtr;
    UInt32 scsiDataLength;
    BytePtr scsiSensePtr;
    UInt8 scsiSenseLength;
    UInt8 scsiCDBLength;
    UInt16 scsiSGListCount;
    UInt32 scsiReserved4;
    UInt8 scsiSCSIstatus;
    SInt8 scsiSenseResidual;
    UInt16 scsiReserved5;
    long scsiDataResidual;
    CDB scsiCDB;
    long scsiTimeout;
    BytePtr scsiReserved5pt5;
    UInt16 scsiReserved5pt6;
    UInt16 scsiIOFlags;
    UInt8 scsiTagAction;
    UInt8 scsiReserved6;
    UInt16 scsiReserved7;
    UInt16 scsiSelectTimeout;
    UInt8 scsiDataType;
    UInt8 scsiTransferType;
    UInt32 scsiReserved8;
    UInt32 scsiReserved9;
    UInt16 scsiHandshake[8];
    UInt32 scsiReserved10;
    UInt32 scsiReserved11;
    SCSI_IO * scsiCommandLink;
    UInt8 scsiSIMpublics[8];
    UInt8 scsiAppleReserved6[8];
    UInt16 scsiCurrentPhase;
    short scsiSelector;
    OSErr scsiOldCallResult;
    UInt8 scsiSCSImessage;
    UInt8 XPTprivateFlags;
    UInt8 XPTextras[12];
};
typedef struct SCSI_IO SCSI_IO;
typedef SCSI_IO SCSIExecIOPB;
```

Fields

`qLink`
A pointer to the next entry in the request queue. This field is used internally by the SCSI Manager and must be set to 0 when the parameter block is initialized. The SCSI Manager functions always set this field to 0 before returning, so you do not need to set it to 0 again before reusing a parameter block.

`scsiReserved1`
Reserved.

`scsiPBLength`
The size of the parameter block, in bytes, including the parameter block header.

`scsiFunctionCode`
A function selector code that specifies the service being requested.

`scsiReserved2`
Reserved.

`scsiResult`
The result code returned by the XPT or SIM when the function completes. The value `scsiRequestInProgress` indicates that the request is still in progress or queued.

`scsiDevice`
A 4-byte value that uniquely identifies the target device for a request. The `DeviceIdent` data type designates the bus number, target SCSI ID, and logical unit number (LUN).

`scsiCompletion`
A pointer to a completion routine.

`scsiFlags`
Flags indicating the transfer direction and any special handling required for this request.

`scsiDriverStorage`
A pointer to the device driver's private storage. This field is not affected or used by the SCSI Manager.

`scsiXPTprivate`
Private field for use in XPT.

`scsiReserved3`
Reserved.

`scsiResultFlags`
Output flags that modify the `scsiResult` field.

`scsiReserved3pt5`
Reserved.

`scsiDataPtr`
A pointer to a data buffer or scatter/gather list. You specify the data type using the `scsiDataType` field.

`scsiDataLength`
The amount of data to be transferred, in bytes.

`scsiSensePtr`
A pointer to the autosense data buffer. If autosense is enabled (the `scsiDisableAutosense` flag is not set), the SCSI Manager returns `REQUEST SENSE` information in this buffer.

`scsiSenseLength`
The size of the autosense data buffer, in bytes.

`scsiCDBLength`
The length of the SCSI command descriptor block, in bytes.

`scsiSGListCount`
The number of elements in the scatter/gather list.

`scsiReserved4`
Reserved.

`scsiSCSIstatus`
The status returned by the SCSI device.

`scsiSenseResidual`
The automatic `REQUEST SENSE` residual length (that is, the number of bytes that were expected but not transferred). This number is negative if extra bytes had to be transferred to force the target off of the bus.

`scsiReserved5`
Reserved for output.

`scsiDataResidual`
The data transfer residual length (that is, the number of bytes that were expected but not transferred). This number is negative if extra bytes had to be transferred to force the target off the bus.

`scsiCDB`
This field can contain either the actual CDB or a pointer to the CDB. You set the `scsiCDBIsPointer` flag if this field contains a pointer.

`scsiTimeout`
The length of time the SIM should allow before reporting a timeout of the SCSI bus. The time value is represented in Time Manager format (positive values for milliseconds, negative values for microseconds). The timer is started when the I/O request is sent to the target. If the request does not complete within the specified time, the SIM attempts to issue an `ABORT` message, either by reselecting the device or by asserting the attention (/ATN) signal. A value of 0 specifies the default timeout for the SIM. The default timeout for the SCSI Manager 4.3 SIM is infinite (that is, no timeout).

`scsiReserved5pt5`
Reserved.

`scsiReserved5pt6`
Reserved.

`scsiIOFlags`
Additional I/O flags describing the data transfer.

`scsiTagAction`
Reserved.

`scsiReserved6`
Reserved for input.

`scsiReserved7`
Reserved for input.

`scsiSelectTimeout`
An optional SELECT timeout value, in milliseconds. The default is 250 ms, as specified by SCSI-2. The accuracy of this period is dependent on the HBA.A value of 0 specifies the default timeout. Some SIMs ignore this parameter and always use a value of 250 ms.

`scsiDataType`
The data type pointed to by the `scsiDataPtr` field.

`scsiTransferType`
The type of transfer mode to use during the data phase.

`scsiReserved8`
Reserved for input.

`scsiReserved9`
Reserved for input.

`scsiHandshake`
Handshaking instructions for blind transfers, consisting of an array of word values, terminated by 0. The SIM polls for data ready after transferring the amount of data specified in each successive `scsiHandshake` entry. When it encounters a 0 value, the SIM starts over at the beginning of the list. Handshaking always starts from the beginning of the list every time a device transitions to data phase.

`scsiReserved10`
Reserved for input.

`scsiReserved11`
Reserved for input.

`scsiCommandLink`
A pointer to a linked parameter block. This field provides support for SCSI linked commands. This optional feature ensures that a set of commands sent to a device are executed in sequential order without interference from other applications. You create a list of commands using this pointer to link additional parameter blocks. Each parameter block except the last should have the `scsiCDBLinked` flag set in the `scsiFlags` field. A `CHECK CONDITION` status from the device will abort linked command execution. Linked commands may not be supported by all SIMs.

`scsiSIMpublics`
An additional input field available for use by SIM developers.

`scsiCurrentPhase`
The current SCSI bus phase reported by the SIM after handling an original SCSI Manager function. This field is used only by the XPT and SIM during original SCSI Manager emulation.

`scsiSelector`
The function selector code that was passed to the `_SCSIDispatch` trap during original SCSI Manager emulation. The SIM uses this field to determine which original SCSI Manager function to perform.

`scsiOldCallResult`
The result code from an emulated original SCSI Manager function. The SIM returns results to all original SCSI Manager functions in this field, except for the `SCSIComplete` result, which it returns in `scsiResult`.

`scsiSCSImessage`
The message byte returned by an emulated `SCSIComplete` function. This field is only used by the XPT and SIM during original SCSI Manager emulation.

`XPTprivateFlags`
Reserved.

`XPTextras`
Reserved.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`SCSI.h`

SCSIBusInquiryPB

Defines a SCSI bus inquiry parameter block.

```
struct SCSIBusInquiryPB {
    SCSIHdr * qLink;
    short scsiReserved1;
    UInt16 scsiPBLength;
    UInt8 scsiFunctionCode;
    UInt8 scsiReserved2;
    volatile OSErr scsiResult;
    DeviceIdent scsiDevice;
    SCSICallbackUPP scsiCompletion;
    UInt32 scsiFlags;
    BytePtr scsiDriverStorage;
    Ptr scsiXPTprivate;
    long scsiReserved3;
    UInt16 scsiEngineCount;
    UInt16 scsiMaxTransferType;
    UInt32 scsiDataTypes;
    UInt16 scsiIOpbSize;
    UInt16 scsiMaxIOpbSize;
    UInt32 scsiFeatureFlags;
    UInt8 scsiVersionNumber;
    UInt8 scsiHBAInquiry;
    UInt8 scsiTargetModeFlags;
    UInt8 scsiScanFlags;
    UInt32 scsiSIMPrivatesPtr;
    UInt32 scsiSIMPrivatesSize;
    UInt32 scsiAsyncFlags;
    UInt8 scsiHiBusID;
    UInt8 scsiInitiatorID;
    UInt16 scsiBIReserved0;
    UInt32 scsiBIReserved1;
    UInt32 scsiFlagsSupported;
    UInt16 scsiIOFlagsSupported;
    UInt16 scsiWeirdStuff;
    UInt16 scsiMaxTarget;
    UInt16 scsiMaxLUN;
    char scsiSIMVendor[16];
    char scsiHBAVendor[16];
    char scsiControllerFamily[16];
    char scsiControllerType[16];
    char scsiXPTversion[4];
    char scsiSIMversion[4];
    char scsiHBAversion[4];
    UInt8 scsiHBAslotType;
    UInt8 scsiHBAslotNumber;
    UInt16 scsiSIMsRsrcID;
    UInt16 scsiBIReserved3;
    UInt16 scsiAdditionalLength;
};
typedef struct SCSIBusInquiryPB SCSIBusInquiryPB;
```

Fields

`qLink`

A pointer to the next entry in the request queue. This field is used internally by the SCSI Manager and must be set to 0 when the parameter block is initialized. The SCSI Manager functions always set this field to 0 before returning, so you do not need to set it to 0 again before reusing a parameter block.

`scsiReserved1`

Reserved.

`scsiPBLength`
The size of the parameter block, in bytes, including the parameter block header.

`scsiFunctionCode`
A function selector code that specifies the service being requested.

`scsiReserved2`
Reserved.

`scsiResult`
The result code returned by the XPT or SIM when the function completes. The value `scsiRequestInProgress` indicates that the request is still in progress or queued.

`scsiDevice`
A 4-byte value that uniquely identifies the target device for a request. The `DeviceIdent` data type designates the bus number, target SCSI ID, and logical unit number (LUN).

`scsiCompletion`
A pointer to a completion routine.

`scsiFlags`
Flags indicating the transfer direction and any special handling required for this request.

`scsiDriverStorage`
A pointer to the device driver's private storage. This field is not affected or used by the SCSI Manager.

`scsiXPTprivate`
Private field for use in XPT.

`scsiReserved3`
Reserved.

`scsiEngineCount`
The number of engines on the HBA. This value is 0 for a built-in SCSI bus.

`scsiMaxTransferType`
The number of data transfer types available on the HBA.

`scsiDataTypes`
A bit mask describing the data types supported by the SIM/HBA. Bits 3 through 15 and bit 31 are reserved by Apple Computer, Inc. Bits 16 through 30 are available for use by SIM developers.

`scsiIOpbSize`
The minimum size of a SCSI I/O parameter block for this SIM.

`scsiMaxIOpbSize`
The minimum size of a SCSI I/O parameter block for all currently registered SIMs. That is, the largest registered `scsiIOpbSize`.

`scsiFeatureFlags`
These flags describe various physical characteristics of the SCSI bus.

`scsiVersionNumber`
The version number of the SIM/HBA.

`scsiHBAInquiry`
Flags describing the capabilities of the bus.

`scsiTargetModeFlags`
Reserved.

`scsiScanFlags`
Reserved.

`scsiSIMPrivatesPtr`
A pointer to the SIM's private storage.

`scsiSIMPrivatesSize`
The size of the SIM's private storage, in bytes.

`scsiAsyncFlags`
Reserved.

`scsiHiBusID`
The highest bus number currently registered with the XPT. If no buses are registered, this field contains 0xFF (the ID of the XPT).

`scsiInitiatorID`
The SCSI ID of the HBA. This value is 7 for a built-in SCSI bus.

`scsiBIReserved0`
`scsiBIReserved1`
`scsiFlagsSupported`
A bit mask that defines which `scsiFlags` bits are supported.

`scsiIOFlagsSupported`
A bit mask that defines which `scsiIOFlags` bits are supported.

`scsiWeirdStuff`
Flags that identify unusual aspects of a SIM's operation.

`scsiMaxTarget`
The highest SCSI ID value supported by the HBA.

`scsiMaxLUN`
The highest logical unit number supported by the HBA.

`scsiSIMVendor`
An ASCII text string that identifies the SIM vendor. This field returns `'Apple Computer'` for a built-in SCSI bus.

`scsiHBAVendor`
An ASCII text string that identifies the HBA vendor. This field returns `'Apple Computer'` for a built-in SCSI bus.

`scsiControllerFamily`
An optional ASCII text string that identifies the family of parts to which the SCSI controller chip belongs. This information is provided at the discretion of the HBA vendor.

`scsiControllerType`
An optional ASCII text string that identifies the specific type of SCSI controller chip. This information is provided at the discretion of the HBA vendor.

`scsiXPTversion`
An ASCII text string that identifies the version number of the XPT. You should use the other fields of this parameter block to check for specific features, rather than relying on this value.

`scsiSIMversion`
An ASCII text string that identifies the version number of the SIM. You should use the other fields of this parameter block to check for specific features, rather than relying on this value.

`scsiHBAversion`
An ASCII text string that identifies the version number of the HBA. You should use the other fields of this parameter block to check for specific features, rather than relying on this value.

`scsiHBAslotType`
The slot type, if any, used by this HBA.

`scsiHBAslotNumber`
: The slot number for the SIM. Device drivers should copy this value into the `dCtlSlot` field of the device control entry. This value is 0 for a built-in SCSI bus.

`scsiSIMsRsrcID`
: The ID for the SIM. Device drivers should copy this value into the `dCtlSlotID` field of the device control entry. This value is 0 for a built-in SCSI bus.

`scsiAdditionalLength`
: The additional size of this parameter block, in bytes. If this structure includes extra fields to return additional information, this field contains the number of additional bytes.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`SCSI.h`

SCSIAbortCommandPB

Defines a SCSI abort command parameter block.

```
struct SCSIAbortCommandPB {
    SCSIHdr * qLink;
    short scsiReserved1;
    UInt16 scsiPBLength;
    UInt8 scsiFunctionCode;
    UInt8 scsiReserved2;
    volatile OSErr scsiResult;
    DeviceIdent scsiDevice;
    SCSICallbackUPP scsiCompletion;
    UInt32 scsiFlags;
    BytePtr scsiDriverStorage;
    Ptr scsiXPTprivate;
    long scsiReserved3;
    SCSI_IO * scsiIOptr;
};
typedef struct SCSIAbortCommandPB SCSIAbortCommandPB;
```

Fields
`qLink`
: A pointer to the next entry in the request queue. This field is used internally by the SCSI Manager and must be set to 0 when the parameter block is initialized. The SCSI Manager functions always set this field to 0 before returning, so you do not need to set it to 0 again before reusing a parameter block.

`scsiReserved1`
: Reserved.

`scsiPBLength`
: The size of the parameter block, in bytes, including the parameter block header.

`scsiFunctionCode`
: A function selector code that specifies the service being requested.

`scsiReserved2`
: Reserved.

`scsiResult`
: The result code returned by the XPT or SIM when the function completes. The value `scsiRequestInProgress` indicates that the request is still in progress or queued.

`scsiDevice`
: A 4-byte value that uniquely identifies the target device for a request. The `DeviceIdent` data type designates the bus number, target SCSI ID, and logical unit number (LUN).

`scsiCompletion`
: A pointer to a completion routine.

`scsiFlags`
: Flags indicating the transfer direction and any special handling required for this request.

`scsiDriverStorage`
: A pointer to the device driver's private storage. This field is not affected or used by the SCSI Manager.

`scsiXPTprivate`
: Private field for use in XPT.

`scsiReserved3`
: Reserved.

`scsiIOptr`
: A pointer to the parameter block to be canceled.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`SCSI.h`

SCSITerminateIOPB

Defines a SCSI terminate I/O parameter block.

```
struct SCSITerminateIOPB {
    SCSIHdr * qLink;
    short scsiReserved1;
    UInt16 scsiPBLength;
    UInt8 scsiFunctionCode;
    UInt8 scsiReserved2;
    volatile OSErr scsiResult;
    DeviceIdent scsiDevice;
    SCSICallbackUPP scsiCompletion;
    UInt32 scsiFlags;
    BytePtr scsiDriverStorage;
    Ptr scsiXPTprivate;
    long scsiReserved3;
    SCSI_IO * scsiIOptr;
};
typedef struct SCSITerminateIOPB SCSITerminateIOPB;
```

Fields

`qLink`
: A pointer to the next entry in the request queue. This field is used internally by the SCSI Manager and must be set to 0 when the parameter block is initialized. The SCSI Manager functions always set this field to 0 before returning, so you do not need to set it to 0 again before reusing a parameter block.

`scsiReserved1`
: Reserved.

`scsiPBLength`
: The size of the parameter block, in bytes, including the parameter block header.

`scsiFunctionCode`
: A function selector code that specifies the service being requested.

`scsiReserved2`
: Reserved.

`scsiResult`
: The result code returned by the XPT or SIM when the function completes. The value `scsiRequestInProgress` indicates that the request is still in progress or queued.

`scsiDevice`
: A 4-byte value that uniquely identifies the target device for a request. The `DeviceIdent` data type designates the bus number, target SCSI ID, and logical unit number (LUN).

`scsiCompletion`
: A pointer to a completion routine.

`scsiFlags`
: Flags indicating the transfer direction and any special handling required for this request.

`scsiDriverStorage`
: A pointer to the device driver's private storage. This field is not affected or used by the SCSI Manager.

`scsiXPTprivate`
: Private field for use in XPT.

`scsiReserved3`
: Reserved.

`scsiIOptr`
: A pointer to the parameter block to be canceled.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`SCSI.h`

SCSIGetVirtualIDInfoPB

Defines a SCSI virtual ID information parameter block.

```
struct SCSIGetVirtualIDInfoPB {
    SCSIHdr * qLink;
    short scsiReserved1;
    UInt16 scsiPBLength;
    UInt8 scsiFunctionCode;
    UInt8 scsiReserved2;
    volatile OSErr scsiResult;
    DeviceIdent scsiDevice;
    SCSICallbackUPP scsiCompletion;
    UInt32 scsiFlags;
    Ptr scsiDriverStorage;
    Ptr scsiXPTprivate;
    long scsiReserved3;
    UInt16 scsiOldCallID;
    Boolean scsiExists;
    SInt8 filler;
};
typedef struct SCSIGetVirtualIDInfoPB SCSIGetVirtualIDInfoPB;
```

Fields

`qLink`
A pointer to the next entry in the request queue. This field is used internally by the SCSI Manager and must be set to 0 when the parameter block is initialized. The SCSI Manager functions always set this field to 0 before returning, so you do not need to set it to 0 again before reusing a parameter block.

`scsiReserved1`
Reserved.

`scsiPBLength`
The size of the parameter block, in bytes, including the parameter block header.

`scsiFunctionCode`
A function selector code that specifies the service being requested.

`scsiReserved2`
Reserved.

`scsiResult`
The result code returned by the XPT or SIM when the function completes. The value `scsiRequestInProgress` indicates that the request is still in progress or queued.

`scsiDevice`
A 4-byte value that uniquely identifies the target device for a request. The `DeviceIdent` data type designates the bus number, target SCSI ID, and logical unit number (LUN).

`scsiCompletion`
A pointer to a completion routine.

`scsiFlags`
: Flags indicating the transfer direction and any special handling required for this request.

`scsiDriverStorage`
: A pointer to the device driver's private storage. This field is not affected or used by the SCSI Manager.

`scsiXPTprivate`
: Private field for use in XPT.

`scsiReserved3`
: Reserved.

`scsiOldCallID`
: The virtual SCSI ID of the device you are searching for.

`scsiExists`
: The XPT returns true in this field if the `scsiDevice` field contains a valid device identification record.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`SCSI.h`

SCSILoadDriverPB

Defines a SCSI load driver parameter block.

```
struct SCSILoadDriverPB {
    SCSIHdr * qLink;
    short scsiReserved1;
    UInt16 scsiPBLength;
    UInt8 scsiFunctionCode;
    UInt8 scsiReserved2;
    volatile OSErr scsiResult;
    DeviceIdent scsiDevice;
    SCSICallbackUPP scsiCompletion;
    UInt32 scsiFlags;
    Ptr scsiDriverStorage;
    Ptr scsiXPTprivate;
    long scsiReserved3;
    short scsiLoadedRefNum;
    Boolean scsiDiskLoadFailed;
    SInt8 filler;
};
typedef struct SCSILoadDriverPB SCSILoadDriverPB;
```

Fields
`qLink`
: A pointer to the next entry in the request queue. This field is used internally by the SCSI Manager and must be set to 0 when the parameter block is initialized. The SCSI Manager functions always set this field to 0 before returning, so you do not need to set it to 0 again before reusing a parameter block.

`scsiReserved1`
: Reserved.

`scsiPBLength`
: The size of the parameter block, in bytes, including the parameter block header.

`scsiFunctionCode`
: A function selector code that specifies the service being requested.

`scsiReserved2`
: Reserved.

`scsiResult`
: The result code returned by the XPT or SIM when the function completes. The value `scsiRequestInProgress` indicates that the request is still in progress or queued.

`scsiDevice`
: A 4-byte value that uniquely identifies the target device for a request. The `DeviceIdent` data type designates the bus number, target SCSI ID, and logical unit number (LUN).

`scsiCompletion`
: A pointer to a completion routine.

`scsiFlags`
: Flags indicating the transfer direction and any special handling required for this request.

`scsiDriverStorage`
: A pointer to the device driver's private storage. This field is not affected or used by the SCSI Manager.

`scsiXPTprivate`
: Private field for use in XPT.

`scsiReserved3`
: Reserved.

`scsiLoadedRefNum`
: If the driver is successfully loaded, this field contains the driver reference number returned by the SIM.

`scsiDiskLoadFailed`
: If this field is set to true, the SIM should attempt to load its own driver regardless of whether there is one on the device. If this field is set to false, the SIM has the option of loading a driver from the device or using one of its own.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`SCSI.h`

SCSIDriverPB

Defines a SCSI driver identification parameter block.

```
struct SCSIDriverPB {
    SCSIHdr * qLink;
    short scsiReserved1;
    UInt16 scsiPBLength;
    UInt8 scsiFunctionCode;
    UInt8 scsiReserved2;
    volatile OSErr scsiResult;
    DeviceIdent scsiDevice;
    SCSICallbackUPP scsiCompletion;
    UInt32 scsiFlags;
    Ptr scsiDriverStorage;
    Ptr scsiXPTprivate;
    long scsiReserved3;
    short scsiDriver;
    UInt16 scsiDriverFlags;
    DeviceIdent scsiNextDevice;
};
typedef struct SCSIDriverPB SCSIDriverPB;
```

Fields

`qLink`
A pointer to the next entry in the request queue. This field is used internally by the SCSI Manager and must be set to 0 when the parameter block is initialized. The SCSI Manager functions always set this field to 0 before returning, so you do not need to set it to 0 again before reusing a parameter block.

`scsiReserved1`
Reserved.

`scsiPBLength`
The size of the parameter block, in bytes, including the parameter block header.

`scsiFunctionCode`
A function selector code that specifies the service being requested.

`scsiReserved2`
Reserved.

`scsiResult`
The result code returned by the XPT or SIM when the function completes. The value `scsiRequestInProgress` indicates that the request is still in progress or queued.

`scsiDevice`
A 4-byte value that uniquely identifies the target device for a request. The `DeviceIdent` data type designates the bus number, target SCSI ID, and logical unit number (LUN).

`scsiCompletion`
A pointer to a completion routine.

`scsiFlags`
Flags indicating the transfer direction and any special handling required for this request.

`scsiDriverStorage`
A pointer to the device driver's private storage. This field is not affected or used by the SCSI Manager.

`scsiXPTprivate`
`scsiReserved3`
Reserved.

`scsiDriver`
The driver reference number of the device driver associated with this device identification record.

`scsiDriverFlags`
: Driver information flags. These flags are not interpreted by the XPT but can be used to provide information about the driver to other clients.

`scsiNextDevice`
: The device identification record of the next device in the driver registration list.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`SCSI.h`

Constants

SCSI Flags

Used in the `scsiFlags` field of the `SCSI_PB` structure.

```
enum {
    scsiDirectionMask = 0xC0000000,
    scsiDirectionNone = 0xC0000000,
    scsiDirectionReserved = 0x00000000,
    scsiDirectionOut = 0x80000000,
    scsiDirectionIn = 0x40000000,
    scsiDisableAutosense = 0x20000000,
    scsiFlagReservedA = 0x10000000,
    scsiFlagReserved0 = 0x08000000,
    scsiCDBLinked = 0x04000000,
    scsiQEnable = 0x02000000,
    scsiCDBIsPointer = 0x01000000,
    scsiFlagReserved1 = 0x00800000,
    scsiInitiateSyncData = 0x00400000,
    scsiDisableSyncData = 0x00200000,
    scsiSIMQHead = 0x00100000,
    scsiSIMQFreeze = 0x00080000,
    scsiSIMQNoFreeze = 0x00040000,
    scsiDoDisconnect = 0x00020000,
    scsiDontDisconnect = 0x00010000,
    scsiDataReadyForDMA = 0x00008000,
    scsiFlagReserved3 = 0x00004000,
    scsiDataPhysical = 0x00002000,
    scsiSensePhysical = 0x00001000,
    scsiFlagReserved5 = 0x00000800,
    scsiFlagReserved6 = 0x00000400,
    scsiFlagReserved7 = 0x00000200,
    scsiFlagReserved8 = 0x00000100
};
```

Constants

`scsiDirectionMask`

A bit field that specifies transfer direction, using these constants: `scsiDirectionIn`, `scsiDirectionOut`, and `scsiDirectionNone`

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiDirectionNone`

No data phase expected.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiDirectionOut`

Data out.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiDirectionIn`
Data in.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiDisableAutosense`
Disable the automatic REQUEST SENSE feature.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiCDBLinked`
The parameter block contains a linked CDB. This option may not be supported by all SIMs.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiQEnable`
Enable target queue actions. This option may not be supported by all SIMs.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiCDBIsPointer`
Set if the `scsiCDB` field of a SCSI I/O parameter block contains a pointer. If clear, the `scsiCDB` field contains the actual CDB. In either case, the `scsiCDBLength` field contains the number of bytes in the SCSI command descriptor block.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiInitiateSyncData`
Set if the SIM should attempt to initiate a synchronous data transfer by sending the SDTR message. If successful, the device normally remains in the synchronous transfer mode until it is reset or until you specify asynchronous mode by setting the `scsiDisableSyncData` flag. Because SDTR negotiation occurs every time this flag is set, you should set it only when negotiation is actually needed.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiDisableSyncData`
Disable synchronous data transfer. The SIM sends an SDTR message with a REQ/ACK offset of 0 to indicate asynchronous data transfer mode. You should set this flag only when negotiation is actually needed.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiSIMQHead`
: Place the parameter block at the head of the SIM queue. This can be used to insert error handling at the head of a frozen queue.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiSIMQFreeze`
: Freeze the SIM queue after completing this transaction.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiSIMQNoFreeze`
: Disable SIM queue freezing for this transaction.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiDoDisconnect`
: Explicitly allow device to disconnect.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiDontDisconnect`
: Explicitly prohibit device disconnection. If this flag and the `scsiDoDisconnect` flag are both 0, the SIM determines whether to allow or prohibit disconnection, based on performance criteria.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiDataReadyForDMA`
: Data buffer is locked and non-cacheable.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiDataPhysical`
: Data buffer address is physical.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiSensePhysical`
: Autosense data pointer is physical.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

SCSIAction function selector codes

Used in the `scsiFunctionCode` field of the parameter block passed to the `SCSIAction` function.

```
enum {
    SCSINop = 0x00,
    SCSIExecIO = 0x01,
    SCSIBusInquiry = 0x03,
    SCSIReleaseQ = 0x04,
    SCSIAbortCommand = 0x10,
    SCSIResetBus = 0x11,
    SCSIResetDevice = 0x12,
    SCSITerminateIO = 0x13
};
enum {
    SCSIGetVirtualIDInfo = 128,
    SCSILoadDriver = 130,
    SCSIOldCall = 132,
    SCSICreateRefNumXref = 133,
    SCSILookupRefNumXref = 134,
    SCSIRemoveRefNumXref = 135,
    SCSIRegisterWithNewXPT = 136
};
```

Constants

`SCSINop`
No operation.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`SCSIExecIO`
Execute a SCSI I/O transaction.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`SCSIBusInquiry`
Bus inquiry.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`SCSIReleaseQ`
Release a frozen SIM queue.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`SCSIAbortCommand`
: Abort a SCSI command.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `SCSI.h`.

`SCSIResetBus`
: Reset the SCSI bus.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `SCSI.h`.

`SCSIResetDevice`
: Reset a SCSI device.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `SCSI.h`.

`SCSITerminateIO`
: Terminate I/O transaction.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `SCSI.h`.

`SCSIGetVirtualIDInfo`
: Return `DeviceIdent` of a virtual SCSI ID.

`SCSILoadDriver`
: Load a driver from a SCSI device.

`SCSIOldCall`
: SIM support function for original SCSI Manager emulation.

`SCSICreateRefNumXref`
: Register a device driver.

`SCSILookupRefNumXref`
: Find a driver reference number.

`SCSIRemoveRefNumXref`
: Deregister a device driver.

`SCSIRegisterWithNewXPT`
: XPT was replaced; SIM needs to reregister.

kBusTypeSCSI

Used in the `diReserved` field of the `DeviceIdent` structure to identify the type of device described by the structure.

```
enum {
    kBusTypeSCSI = 0,
    kBusTypeATA = 1,
    kBusTypePCMCIA = 2,
    kBusTypeMediaBay = 3
};
```

Constants

`kBusTypeSCSI`
`DeviceIdent` holds information about a SCSI device.
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`kBusTypeATA`
`DeviceIdent` holds information about an ATA device.
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`kBusTypePCMCIA`
Not recommended.
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`kBusTypeMediaBay`
Not recommended.
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

SCSI Result Flags

Used in `scsiResultFlags` field of the `SCSI_IO` structure.

```
enum {
    scsiSIMQFrozen = 0x0001,
    scsiAutosenseValid = 0x0002,
    scsiBusNotFree = 0x0004
};
```

Constants

`scsiSIMQFrozen`
The SIM queue for this LUN is frozen because of an error. You must call the `SCSIReleaseQ` function to release the queue and resume processing requests.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`scsiAutosenseValid`
: An automatic `REQUEST SENSE` was performed after this I/O because of a `CHECK CONDITION` status message from the device. The data contained in the `scsiSensePtr` buffer is valid.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `SCSI.h`.

`scsiBusNotFree`
: The SCSI Manager was unable to clear the bus after an error. You may need to call the `SCSIResetBus` function to restore operation.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `SCSI.h`.

SCSI IO Flags

Used in the `scsiIOFlags` field of the `SCSI_IO` structure.

```
enum {
    scsiNoParityCheck = 0x0002,
    scsiDisableSelectWAtn = 0x0004,
    scsiSavePtrOnDisconnect = 0x0008,
    scsiNoBucketIn = 0x0010,
    scsiNoBucketOut = 0x0020,
    scsiDisableWide = 0x0040,
    scsiInitiateWide = 0x0080,
    scsiRenegotiateSense = 0x0100,
    scsiDisableDiscipline = 0x0200,
    scsiIOFlagReserved0080 = 0x0080,
    scsiIOFlagReserved8000 = 0x8000
};
```

Constants

`scsiNoParityCheck`
: Disable parity error detection for this transaction.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `SCSI.h`.

`scsiDisableSelectWAtn`
: Do not send the IDENTIFY message for LUN selection. The LUN is still required in the `scsiDevice` field so that the request can be placed in the proper queue. The LUN field in the CDB is untouched. The purpose is to provide compatibility with older devices that do not support this aspect of the SCSI-2 specification.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `SCSI.h`.

`scsiSavePtrOnDisconnect`

Perform a SAVE DATA POINTER operation automatically in response to a DISCONNECT message from the target. The purpose of this flag is to provide compatibility with devices that do not properly implement this aspect of the SCSI-2 specification.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiNoBucketIn`

Prohibit bit-bucketing during the data-in phase of the transaction. Bit-bucketing is the practice of throwing away excess data bytes when a target tries to supply more data than the initiator expects. For example, if the CDB requests more data than you specified in the `scsiDataLength` field, the SCSI Manager normally throws away the excess and returns the `scsiDataRunError` result code. If this flag is set, the SCSI Manager refuses any extra data, terminates the I/O request, and leaves the bus in the data-in phase. You must reset the bus to restore operation. This flag is intended only for debugging purposes.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiNoBucketOut`

Prohibit bit-bucketing during the data-out phase of the transaction. If a target requests more data than you specified in the `scsiDataLength` field, the SCSI Manager normally sends an arbitrary number of meaningless bytes (0xEE) until the target releases the bus. If this flag is set, the SCSI Manager terminates the I/O request when the last byte is sent and leaves the bus in the data-out phase. You must reset the bus to restore operation. This flag is intended only for debugging purposes.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiDisableWide`

Disable wide data transfer negotiation for this transaction if it had been previously enabled. This option may not be supported by all SIMs.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiInitiateWide`

Attempt wide data transfer negotiation for this transaction if it is not already enabled. This option may not be supported by all SIMs.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiRenegotiateSense`
Attempt to renegotiate synchronous or wide transfers before issuing a REQUEST SENSE. This is necessary when the error was caused by problems operating in synchronous or wide transfer mode. It is optional because some devices flush sense data after performing negotiation.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

SCSI_IO Data Types

Used in the `scsiDataType` field of the `SCSI_IO` parameter block.

```
enum {
    scsiDataBuffer = 0,
    scsiDataTIB = 1,
    scsiDataSG = 2,
    scsiDataIOTable = 3
};
```

Constants

`scsiDataBuffer`
The `scsiDataPtr` field contains a pointer to a contiguous data buffer, and the `scsiDataLength` field contains the length of the buffer, in bytes.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiDataTIB`
The `scsiDataPtr` field contains a pointer to a transfer instruction block. This is used by the XPT during original SCSI Manager emulation, when communicating with a SIM that supports this.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiDataSG`
The `scsiDataPtr` field contains a pointer to a scatter/gather list. The `scsiDataLength` field contains the total number of bytes to be transferred, and the `scsiSGListCount` field contains the number of elements in the scatter/gather list.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

SCSIBusInquiryPB Data Types

Used in the `scsiDataTypes` field of the `SCSIBusInquiryPB` structure.

```
enum {
    scsiBusDataTIB = (1 << scsiDataTIB),
    scsiBusDataBuffer = (1 << scsiDataBuffer),
    scsiBusDataSG = (1 << scsiDataSG),
    scsiBusDataIOTable = (1 << scsiDataIOTable),
    scsiBusDataReserved = 0x80000000
};
```

Discussion
These types correspond to the `scsiDataType` field of the SCSI I/O parameter block.

SCSI Transfer Types

Used in the `scsiTransferType` field of the `SCSI_IO` structure.

```
enum {
    scsiTransferBlind = 0,
    scsiTransferPolled = 1
};
```

Constants

`scsiTransferBlind`
Use DMA, if available; otherwise, perform a blind transfer using the handshaking information contained in the `scsiHandshake` field.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`scsiTransferPolled`
Use polled transfer mode. The `scsiHandshake` field is not required for this mode.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

SCSIBusInquiryPB Feature Flags

Used in the `featureFlags` field of the `SCSIBusInquiryPB` structure.

```
enum {
    scsiBusLVD = 0x00000400,
    scsiBusUltra3SCSI = 0x00000200,
    scsiBusUltra2SCSI = 0x00000100,
    scsiBusInternalExternalMask = 0x000000C0,
    scsiBusInternalExternalUnknown = 0x00000000,
    scsiBusInternalExternal = 0x000000C0,
    scsiBusInternal = 0x00000080,
    scsiBusExternal = 0x00000040,
    scsiBusCacheCoherentDMA = 0x00000020,
    scsiBusOldCallCapable = 0x00000010,
    scsiBusUltraSCSI = 0x00000008,
    scsiBusDifferential = 0x00000004,
    scsiBusFastSCSI = 0x00000002,
    scsiBusDMAavailable = 0x00000001
};
```

Constants

`scsiBusInternalExternalUnknown`
The internal/external state of the bus is unknown.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`scsiBusInternalExternal`
The bus is both internal and external.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`scsiBusInternal`
The bus is at least partly internal to the computer.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`scsiBusExternal`
The bus extends outside of the computer.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`scsiBusCacheCoherentDMA`
DMA is cache coherent.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`scsiBusOldCallCapable`
The SIM supports the original SCSI Manager interface.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiBusDifferential`
The bus uses a differential SCSI interface.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiBusFastSCSI`
The bus supports SCSI-2 fast data transfers.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiBusDMAavailables`
DMA is available.

scsiBusMDP

Used in the `scsiHBAInquiry` field of the `SCSIBusInquiryPB` parameter block.

```
enum {
    scsiBusMDP = 0x80,
    scsiBusWide32 = 0x40,
    scsiBusWide16 = 0x20,
    scsiBusSDTR = 0x10,
    scsiBusLinkedCDB = 0x08,
    scsiBusTagQ = 0x02,
    scsiBusSoftReset = 0x01
};
```

Constants

`scsiBusMDP`
Supports the `MODIFY DATA POINTER` message.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiBusWide32`
Supports 32-bit wide transfers.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiBusWide16`
Supports 16-bit wide transfers.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`scsiBusSDTR`
Supports synchronous transfers.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`scsiBusLinkedCDB`
Supports linked commands.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`scsiBusTagQ`
Supports tagged queuing.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`scsiBusSoftReset`
Supports soft reset.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

scsiOddDisconnectUnsafeRead1

Used in the `scsiWeirdStuff` field of the `SCSIBusInquiryPB` parameter block.

```
enum {
    scsiOddDisconnectUnsafeRead1 = 0x0001,
    scsiOddDisconnectUnsafeWrite1 = 0x0002,
    scsiBusErrorsUnsafe = 0x0004,
    scsiRequiresHandshake = 0x0008,
    scsiTargetDrivenSDTRSafe = 0x0010,
    scsiOddCountForPhysicalUnsafe = 0x0020,
    scsiAbortCmdFixed = 0x0040,
    scsiMeshACKTimingFixed = 0x0080
};
```

Constants

`scsiOddDisconnectUnsafeRead1`
Indicates that a disconnect or other phase change on a odd byte boundary during a read operation will result in inaccurate residual counts or data loss. If your device can disconnect on odd bytes, use polled transfers instead of blind.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiOddDisconnectUnsafeWrite1`
Indicates that a disconnect or other phase change on a odd byte boundary during a write operation will result in inaccurate residual counts or data loss. If your device can disconnect on odd bytes, use polled transfers instead of blind.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiBusErrorsUnsafe`
Indicates that a delay of more than 16 microseconds or a phase change during a blind transfer on a non-handshaked boundary may cause a system crash. If you cannot predict where delays or disconnects will occur, use polled transfers.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiRequiresHandshake`
Indicates that a delay of more than 16 microseconds or a phase change during a blind transfer on a non-handshaked boundary may result in inaccurate residual counts or data loss. If you cannot predict where delays or disconnects will occur, use polled transfers.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiTargetDrivenSDTRSafe`
Indicates that the SIM supports target-initiated synchronous data transfer negotiation. If your device supports this feature and this bit is not set, you must set the `scsiDisableSelectWAtn` flag in the `scsiIOFlags` field.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

scsiMotherboardBus

Used in the `scsiHBAslotType` field of the `SCSIBusInquiryPB` parameter block.

```
enum {
    scsiMotherboardBus = 0x00,
    scsiNuBus = 0x01,
    scsiPDSBus = 0x03,
    scsiPCIBus = 0x04,
    scsiPCMCIABus = 0x05,
    scsiFireWireBridgeBus = 0x06,
    scsiUSBBus = 0x07
};
```

Constants

`scsiMotherboardBus`
A built-in SCSI bus.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`scsiNuBus`
A NuBus slot.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`scsiPDSBus`
A processor-direct slot.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`scsiPCIBus`
A SIM on a PCI bus card.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`scsiPCMCIABus`
A SIM on a PCMCIA card.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`scsiFireWireBridgeBus`
A SIM connected through a FireWire bridge.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `SCSI.h`.

`scsiUSBBus`

A SIM connected on a USB bus.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

kDataOutPhase

Used in the `scsiCurrentPhase` field of the `SCSI_IO` structure.

```
enum {
    kDataOutPhase = ,
    kDataInPhase = 1,
    kCommandPhase = 2,
    kStatusPhase = 3,
    kPhaseIllegal0 = 4,
    kPhaseIllegal1 = 5,
    kMessageOutPhase = 6,
    kMessageInPhase = 7,
    kBusFreePhase = 8,
    kArbitratePhase = 9,
    kSelectPhase = 10,
    kMessageInPhaseNACK = 11
};
```

scsiErrorBase

```
enum {
    scsiErrorBase = -7936
};
```

scsiExecutionErrors

```
enum {
    scsiExecutionErrors = scsiErrorBase,
    scsiNotExecutedErrors = scsiTooManyBuses,
    scsiParameterErrors = scsiPBLengthError
};
```

scsiVERSION

```
enum {
    scsiVERSION = 43
};
```

vendorUnique

```
enum {
    vendorUnique = 0xC0
};
```

scsiDeviceSensitive

Used in the `scsiDriverFlags` field of the `SCSIDriverPB` parameter block.

```
enum {
    scsiDeviceSensitive = 0x0001,
    scsiDeviceNoOldCallAccess = 0x0002
};
```

Constants

`scsiDeviceSensitive`

Only the device driver should access this device. SCSI utilities and other applications that bypass drivers should check this flag before accessing a device.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

`scsiDeviceNoOldCallAccess`

This driver or device does not accept original SCSI Manager requests.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `SCSI.h`.

Result Codes

The table below shows the result codes most commonly returned by the SCSI Manager.

Result Code	Value	Description
`scsiRequestInProgress`	1	Parameter block request is in progress Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scCommErr`	2	Communications error, operation timeout.
`scArbNBErr`	3	Bus busy, arbitration timeout.
`scBadParmsErr`	4	Unrecognized TIB instruction.
`scPhaseErr`	5	Phase error on the SCSI bus.
`scCompareErr`	6	Comparison error from `scComp` instruction.
`scMgrBusyErr`	7	SCSI Manager busy.
`scSequenceErr`	8	Attempted operation is out of sequence.
`scBusTOErr`	9	Bus timeout during blind transfer.
`scComplPhaseErr`	10	SCSI bus was not in status phase on entry to `SCSIComplete`.

Result Code	Value	Description
`scsiPluginInternalError`	-7848	Internal consistency check failed Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiFamilyInternalError`	-7849	Internal consistency check failed Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiCannotLoadPlugin`	-7849	No matching service category Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiBadConnType`	-7850	Bad connection type Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiTargetReserved`	-7853	Target already reserved Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiIOInProgress`	-7854	Can't close connection, I/O in progress Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiBadConnID`	-7856	Bad connection ID Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiInvalidMsgType`	-7858	Invalid message type Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiPartialPrepared`	-7859	Could not do full prepare mem for I/O Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiBadDataLength`	-7860	A zero data length in the parameter block. Available in Mac OS X v10.0 and later. Not available to 64-bit applications.

Result Code	Value	Description
`scsiCDBLengthInvalid`	-7863	The CDB length supplied is not supported by this SIM; typically this means it was too big Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiTransferTypeInvalid`	-7864	The `scsiTransferType` requested is not supported by this SIM Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiDataTypeInvalid`	-7865	SIM does not support the requested `scsiDataType` Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiIDInvalid`	-7866	The initiator ID is invalid Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiLUNInvalid`	-7867	The logical unit number is invalid Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiTIDInvalid`	-7868	The target ID is invalid Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiBusInvalid`	-7869	The bus ID is invalid Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiRequestInvalid`	-7870	The parameter block request is invalid Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiFunctionNotAvailable`	-7871	The requested function is not supported by this SIM Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiPBLengthError`	-7872	The parameter block length supplied was too small for this SIM Available in Mac OS X v10.0 and later. Not available to 64-bit applications.

Result Code	Value	Description
scsiQLinkInvalid	-7881	The qLink field was not 0 Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
scsiNoSuchXref	-7882	No driver has been cross-referenced with this device Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
scsiDeviceConflict	-7883	Attempt to register more than one driver to a device Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
scsiNoHBA	-7884	No HBA detected Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
scsiDeviceNotThere	-7885	SCSI device not installed or available Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
scsiProvideFail	-7886	Unable to provide the requested service Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
scsiBusy	-7887	SCSI subsystem is busy Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
scsiTooManyBuses	-7888	SIM registration failed because the XPT registry is full Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
scsiCDBReceived	-7910	The SCSI CDB was received Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
scsiNoNexus	-7911	Nexus is not established Available in Mac OS X v10.0 and later. Not available to 64-bit applications.

Result Code	Value	Description
`scsiTerminated`	-7912	Parameter block request terminated by the host Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiBDRsent`	-7913	A SCSI bus device reset (BDR) message was sent to the target Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiWrongDirection`	-7915	Data phase was in an unexpected direction Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiSequenceFailed`	-7916	Target bus phase sequence failure Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiUnexpectedBusFree`	-7917	Unexpected bus free phase Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiDataRunError`	-7918	Data overrun/underrun error Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiAutosenseFailed`	-7920	Automatic `REQUEST SENSE` command failed Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiParityError`	-7921	An uncorrectable parity error occurred Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiSCSIBusReset`	-7922	Execution of this parameter block was halted because of a SCSI bus reset Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiMessageRejectReceived`	-7923	`REJECT` message received Available in Mac OS X v10.0 and later. Not available to 64-bit applications.

Result Code	Value	Description
`scsiIdentifyMessageRejected`	-7924	The target issued a `REJECT` message in response to the `IDENTIFY` message; the LUN probably does not exist Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiCommandTimeout`	-7925	The timeout value for this parameter block was exceeded and the parameter block was aborted Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiSelectTimeout`	-7926	Target selection timeout Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiUnableToTerminate`	-7927	Unable to terminate I/O parameter block request Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiNonZeroStatus`	-7932	The target returned non-zero status upon completion of the request Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiUnableToAbort`	-7933	Unable to abort parameter block request Available in Mac OS X v10.0 and later. Not available to 64-bit applications.
`scsiRequestAborted`	-7934	Parameter block request aborted by the host Available in Mac OS X v10.0 and later. Not available to 64-bit applications.

Text Encoding Conversion Manager Reference

Framework: CoreServices/CoreServices.h

Declared in UnicodeConverter.h
TextCommon.h
Unicode.h
TextEncodingConverter.h
TextEncodingPlugin.h

Overview

The Text Encoding Conversion (TEC) Manager provides two facilities—the Text Encoding Converter and the Unicode Converter—that your application can use to handle text encoding conversion on the Mac OS. You will find the Text Encoding Conversion Manager helpful if you develop Internet applications, such as Web browsers or e-mail applications, applications that transfer text across different platforms, or applications based in Unicode.

Functions by Task

Creating a Text Encoding Specification

`CreateTextEncoding` (page 1890)
Creates and returns a text encoding specification.

Obtaining Information From a Text Encoding Specification

`GetTextEncodingBase` (page 1899)
Returns the base encoding of the specified text encoding.

`GetTextEncodingFormat` (page 1899)
Returns the format value of the specified text encoding.

`GetTextEncodingName` (page 1899)
Returns the localized name for a specified text encoding.

`GetTextEncodingVariant` (page 1901)
Returns the variant from the specified text encoding.

`ResolveDefaultTextEncoding` (page 1906)
Returns a text encoding specification in which any meta-values have been resolved to real values. Currently, this affects only the base encoding values packed into the text encoding specification.

Converting Between Script Manager Values and Text Encodings

`RevertTextEncodingToScriptInfo` (page 1907)
Converts the given Mac OS text encoding specification to the corresponding script code and, if possible, language code and font name.

`UpgradeScriptInfoToTextEncoding` (page 1937)
Converts any combination of a Mac OS script code, a language code, a region code, and a font name to a text encoding.

Obtaining Information About Available Text Encodings

`TECCountAvailableTextEncodings` (page 1914)
Counts and returns the number of text encodings currently configured in the Text Encoding Converter.

`TECCountSubTextEncodings` (page 1917)
Counts and returns the number of subencodings a text encoding supports.

`TECGetAvailableTextEncodings` (page 1926)
Returns the text encoding specifications currently configured in the Text Encoding Converter.

`TECGetSubTextEncodings` (page 1930)
Returns the text encoding specifications for the subencodings the encoding scheme supports.

`NearestMacTextEncodings` (page 1902)
Obtains the best and alternate Mac text encoding.

Identifying Direct Encoding Conversions

`TECCountDirectTextEncodingConversions` (page 1916)
Counts and returns the number of direct conversions currently configured in the Text Encoding Converter.

`TECGetDirectTextEncodingConversions` (page 1927)
Returns the types of direct conversions currently configured in the Text Encoding Converter.

Identifying Possible Destination Encodings

`TECCountDestinationTextEncodings` (page 1915)
Counts and returns the number of destination encodings to which a specified source encoding can be converted in one step.

`TECGetDestinationTextEncodings` (page 1926)
Returns the encoding specifications for all the destination text encodings to which the Text Encoding Converter can directly convert the specified source encoding.

Obtaining Converter Information

`TECGetInfo` (page 1929)
Allocates a converter information structure of type `TECInfo` in the application heap using `NewHandle`, fills it out, and returns a handle.

Creating and Deleting Converter Objects

`TECCreateConverter` (page 1918)
Determines a conversion path for a source and destination encoding, then creates a text encoding converter object and returns a pointer to it.

`TECCreateConverterFromPath` (page 1919)
Creates a converter object for a specific conversion path—from a source encoding through intermediate encodings to a destination encoding—and returns a pointer to it.

`TECClearConverterContextInfo` (page 1910)
Resets a converter object to its initial state so you can reuse it.

`TECDisposeConverter` (page 1921)
Disposes of a converter object.

Converting Text Between Encodings

`TECConvertText` (page 1911)
Converts a stream of text from a source encoding to a destination encoding. It uses the conversion path specified by the converter object you supply.

`TECFlushText` (page 1924)
Flushes out any data in a converter object's temporary buffers and resets the converter object.

Converting to Multiple Encoding Runs

`TECConvertTextToMultipleEncodings` (page 1912)
Converts text in the source encoding to runs of text in multiple destination encodings. It uses the conversion path specified in the converter object you supply.

`TECCreateOneToManyConverter` (page 1920)
Determines a conversion path for the source encoding and destinations encodings you specify, creates a text encoding converter object, and returns a reference to it.

`TECFlushMultipleEncodings` (page 1922)
Flushes out any encodings that may be stored in a converter object's temporary buffers and shifts encodings back to their default state, if any.

`TECGetEncodingList` (page 1928)
Gets the list of destination encodings from a converter object.

Using Sniffers to Investigate Encodings

`TECCreateSniffer` (page 1920)
Creates a sniffer object and returns a reference to it.

`TECClearSnifferContextInfo` (page 1910)
Resets a sniffer object to its initial settings so you can reuse it.

`TECDisposeSniffer` (page 1922)
Disposes of a sniffer object.

`TECCountAvailableSniffers` (page 1914)
Counts and returns the number of sniffers available in all installed plug-ins.

`TECGetAvailableSniffers` (page 1925)
Returns the list of sniffers available in all installed plug-ins.

`TECSniffTextEncoding` (page 1933)
Analyzes a text stream and returns the probable encodings in a ranked list, based on an array of possible encodings you supply. It also returns the number of errors and features for each encoding.

Getting Information About Internet and Regional Text Encoding Names

`TECCountMailTextEncodings` (page 1916)
Counts and returns the number of currently supported e-mail encodings for a specified region.

`TECCountWebTextEncodings` (page 1918)
Counts and returns the number of currently supported text encodings for a region code.

`TECGetMailTextEncodings` (page 1929)
Returns the currently supported mail encoding specifications for a region code.

`TECGetTextEncodingFromInternetName` (page 1931)
Returns the Mac OS text encoding specification that corresponds to an Internet encoding name.

`TECGetTextEncodingInternetName` (page 1931)
Returns the Internet encoding name that corresponds to a Mac OS text encoding.

`TECGetWebTextEncodings` (page 1932)
Returns the currently supported text encoding specifications for a region code.

Converting to Unicode

`ChangeTextToUnicodeInfo` (page 1875)
Changes the mapping information for the specified Unicode converter object used to convert text to Unicode to the new mapping you provide.

`ConvertFromTextToUnicode` (page 1877)
Converts a string from any encoding to Unicode.

`CreateTextToUnicodeInfo` (page 1890)
Creates and returns a Unicode converter object containing information required for converting strings from a non-Unicode encoding to Unicode.

`CreateTextToUnicodeInfoByEncoding` (page 1891)
Based on the given text encoding specification, creates and returns a Unicode converter object containing information required for converting strings from the specified non-Unicode encoding to Unicode.

`DisposeTextToUnicodeInfo` (page 1897)
Releases the memory allocated for the specified Unicode converter object.

`ResetTextToUnicodeInfo` (page 1905)
Reinitializes all state information kept by the context objects.

Converting From Unicode

`ChangeUnicodeToTextInfo` (page 1875)
Changes the mapping information contained in the specified Unicode converter object used to convert Unicode text to a non-Unicode encoding.

`ConvertFromUnicodeToText` (page 1883)
Converts a Unicode text string to the destination encoding you specify.

`CreateUnicodeToTextInfo` (page 1892)
Creates and returns a Unicode converter object containing information required for converting strings from Unicode to a non-Unicode encoding.

`CreateUnicodeToTextInfoByEncoding` (page 1893)
Based on the given text encoding specification for the converted text, creates and returns a Unicode converter object containing information required for converting strings from Unicode to the specified non-Unicode encoding.

`DisposeUnicodeToTextInfo` (page 1898)
Releases the memory allocated for the specified Unicode converter object.

`ResetUnicodeToTextInfo` (page 1905)
Reinitializes all state information kept by a Unicode converter object.

Converting From Unicode to Multiple Encodings

`ConvertFromUnicodeToTextRun` (page 1885)
Converts a string from Unicode to one or more encodings.

`ConvertFromUnicodeToScriptCodeRun` (page 1880)
Converts a string from Unicode to one or more scripts.

`CreateUnicodeToTextRunInfo` (page 1894)
Creates and returns a Unicode converter object containing the information required for converting a Unicode text string to strings in one or more non-Unicode encodings.

`CreateUnicodeToTextRunInfoByEncoding` (page 1895)
Based on the given text encoding specifications for the converted text runs, creates and returns a Unicode converter object containing information required for converting strings from Unicode to one or more specified non-Unicode encodings.

`CreateUnicodeToTextRunInfoByScriptCode` (page 1896)
Based on the given script codes for the converted text runs, creates and returns a Unicode converter object containing information required for converting strings from Unicode to one or more specified non-Unicode encodings.

`DisposeUnicodeToTextRunInfo` (page 1898)
Releases the memory allocated for the specified Unicode converter object.

`ResetUnicodeToTextRunInfo` (page 1906)
Reinitializes all state information kept by the context objects in TextRun conversions.

Converting Between Unicode and Pascal Strings

`ConvertFromPStringToUnicode` (page 1876)
Converts a Pascal string in a Mac OS text encoding to a Unicode string.

`ConvertFromUnicodeToPString` (page 1879)
Converts a Unicode string to Pascal in a Mac OS text encoding.

Obtaining Unicode Mapping Information

`CountUnicodeMappings` (page 1889)
Counts available mappings that meet the specified matching criteria.

`QueryUnicodeMappings` (page 1903)
Returns a list of the conversion mappings available on the system that meet specified matching criteria and returns the number of mappings found.

Truncating Strings Before Converting Them

`TruncateForTextToUnicode` (page 1934)
Identifies where your application can safely break a multibyte string to be converted to Unicode so that the string is not broken in the middle of a multibyte character.

`TruncateForUnicodeToText` (page 1935)
Identifies where your application can safely break a Unicode string to be converted to any encoding so that the string is broken in a way that preserves the text element integrity.

Setting the Fallback Handler

`SetFallbackUnicodeToText` (page 1908)
Specifies a fallback handler to be used for converting a Unicode text segment to another encoding when the Unicode Converter cannot convert the text using the mapping table specified by the Unicode converter object.

`SetFallbackUnicodeToTextRun` (page 1909)
Specifies a fallback handler to be used for converting a Unicode text segment to another encoding when the Unicode Converter cannot convert the text using the mapping table specified by a Unicode converter object.

Working With Universal Procedure Pointers

`NewUnicodeToTextFallbackUPP` (page 1903)
Creates a new universal procedure pointer (UPP) to a Unicode-to-text fallback callback.

`DisposeUnicodeToTextFallbackUPP` (page 1897)
Disposes of a a new universal procedure pointer (UPP) to a Unicode-to-text fallback callback.

`InvokeUnicodeToTextFallbackUPP` (page 1901)
Calls your Unicode-to-text fallback callback.

Getting UniChar Property Values

`UCGetCharProperty` (page 1936)
Obtains the value associated with a property type for the specified `UniChar` characters.

Functions

ChangeTextToUnicodeInfo

Changes the mapping information for the specified Unicode converter object used to convert text to Unicode to the new mapping you provide.

```
OSStatus ChangeTextToUnicodeInfo (
   TextToUnicodeInfo ioTextToUnicodeInfo,
   ConstUnicodeMappingPtr iUnicodeMapping
);
```

Parameters

ioTextToUnicodeInfo
The Unicode converter object of type `TextToUnicodeInfo` (page 1966) containing the mapping to be modified. You use the function `CreateTextToUnicodeInfo` (page 1890) to obtain one.

iUnicodeMapping
A structure of type `UnicodeMapping` (page 1967) identifying the new mapping to be used. This is the mapping that replaces the existing mapping in the Unicode converter object.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
The function replaces the mapping table information that currently exists in the Unicode converter object pointed to by the `ioTextToUnicodeInfo` parameter with the information contained in the `UnicodeMapping` structure you supply as the `iUnicodeMapping` parameter.

`ChangeTextToUnicodeInfo` resets the Unicode converter object's fields as necessary.

If an error is returned, the Unicode converter object is invalid.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

ChangeUnicodeToTextInfo

Changes the mapping information contained in the specified Unicode converter object used to convert Unicode text to a non-Unicode encoding.

```
OSStatus ChangeUnicodeToTextInfo (
   UnicodeToTextInfo ioUnicodeToTextInfo,
   ConstUnicodeMappingPtr iUnicodeMapping
);
```

Parameters

ioUnicodeToTextInfo

The Unicode converter object of type `UnicodeToTextInfo` (page 1969) to be modified. You use the function `CreateUnicodeToTextInfo` (page 1892) or `CreateUnicodeToTextInfoByEncoding` (page 1893) to obtain a Unicode converter object of this type.

iUnicodeMapping

The structure of type `UnicodeMapping` (page 1967) to be used. This is the new mapping that replaces the existing mapping in the Unicode converter object.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Discussion

The function replaces the mapping table information that currently exists in the specified Unicode converter object with the information contained in the new Unicode mapping structure you provide.

`ChangeUnicodeToTextInfo` resets the Unicode converter object's fields as necessary. However, it does not initialize or reset the conversion state maintained by the Unicode converter object.

This function is especially useful for converting a string from Unicode if the Unicode string contains characters that require multiple destination encodings and you know the next destination encoding.

For example, you can change the other (destination) encoding of the Unicode mapping structure pointed to by the `iUnicodeMapping` parameter before you call the function `ConvertFromUnicodeToText` (page 1883) to convert the next character or sequence of characters that require a different destination encoding.

If an error is returned, the Unicode converter object is invalid.

Availability

Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`UnicodeConverter.h`

ConvertFromPStringToUnicode

Converts a Pascal string in a Mac OS text encoding to a Unicode string.

```
OSStatus ConvertFromPStringToUnicode (
   TextToUnicodeInfo iTextToUnicodeInfo,
   ConstStr255Param iPascalStr,
   ByteCount iOutputBufLen,
   ByteCount *oUnicodeLen,
   UniChar oUnicodeStr[]
);
```

Parameters

iTextToUnicodeInfo

A Unicode converter object of type `TextToUnicodeInfo` (page 1966) for the Pascal string to be converted. You can use the function `CreateTextToUnicodeInfo` (page 1890) or `CreateTextToUnicodeInfoByEncoding` (page 1891) to create the Unicode converter object.

iPascalStr

The Pascal string to be converted to Unicode.

iOutputBufLen

The length in bytes of the output buffer pointed to by the `oUnicodeStr` parameter. Your application supplies this buffer to hold the returned converted string. The `oUnicodeLen` parameter may return a byte count that is less than this value if the converted string is smaller than the buffer size you allocated.

oUnicodeLen

On return, a pointer to the length in bytes of the converted Unicode string returned in the `oUnicodeStr` parameter.

oUnicodeStr

A pointer to a Unicode character array. On return, this array holds the converted Unicode string.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Discussion

The `ConvertFromPStringToUnicode` function provides an easy and efficient way to convert a short Pascal string to a Unicode string without incurring the overhead associated with the function `ConvertFromTextToUnicode` (page 1877).

If necessary, this function automatically uses fallback characters to map the text elements of the string.

Availability

Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`UnicodeConverter.h`

ConvertFromTextToUnicode

Converts a string from any encoding to Unicode.

```
OSStatus ConvertFromTextToUnicode (
   TextToUnicodeInfo iTextToUnicodeInfo,
   ByteCount iSourceLen,
   ConstLogicalAddress iSourceStr,
   OptionBits iControlFlags,
   ItemCount iOffsetCount,
   const ByteOffset iOffsetArray[],
   ItemCount *oOffsetCount,
   ByteOffset oOffsetArray[],
   ByteCount iOutputBufLen,
   ByteCount *oSourceRead,
   ByteCount *oUnicodeLen,
   UniChar oUnicodeStr[]
);
```

Parameters

iTextToUnicodeInfo

A Unicode converter object of type `TextToUnicodeInfo` containing mapping and state information used for the conversion. The contents of this Unicode converter object are modified by the function. Your application obtains a Unicode converter object using the function `CreateTextToUnicodeInfo` (page 1890).

iSourceLen

The length in bytes of the source string to be converted.

iSourceStr

The address of the source string to be converted.

iControlFlags

Conversion control flags. You can use "Conversion Masks" (page 1972) to set the `iControlFlags` parameter.

iOffsetCount

The number of offsets in the `iOffsetArray` parameter. Your application supplies this value. The number of entries in `iOffsetArray` must be fewer than the number of bytes specified in `iSourceLen`. If you don't want offsets returned to you, specify `0` (zero) for this parameter.

iOffsetArray

An array of type `ByteOffset`. On input, you specify the array that contains an ordered list of significant byte offsets pertaining to the source string. These offsets may identify font or style changes, for example, in the source string. All array entries must be less than the length in bytes specified by the `iSourceLen` parameter. If you don't want offsets returned to your application, specify `NULL` for this parameter and `0` (zero) for `iOffsetCount`.

oOffsetCount

On return, a pointer to the number of offsets that were mapped in the output stream.

oOffsetArray

An array of type `ByteOffset`. On return, this array contains the corresponding new offsets for the Unicode string produced by the converter.

iOutputBufLen

The length in bytes of the output buffer pointed to by the `oUnicodeStr` parameter. Your application supplies this buffer to hold the returned converted string. The `oUnicodeLen` parameter may return a byte count that is less than this value if the converted byte string is smaller than the buffer size you allocated. The relationship between the size of the source string and the Unicode string is complex and depends on the source encoding and the contents of the string.

oSourceRead

On return, a pointer to the number of bytes of the source string that were converted. If the function returns a `kTECUnmappableElementErr` result code, this parameter returns the number of bytes that were converted before the error occurred.

oUnicodeLen

On return, a pointer to the length in bytes of the converted stream.

oUnicodeStr

A pointer to an array used to hold a Unicode string. On input, this value points to the beginning of the array for the converted string. On return, this buffer holds the converted Unicode string. (For guidelines on estimating the size of the buffer needed, see the discussion.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026). The function returns a `noErr` result code if it has completely converted the input string to Unicode without using fallback characters.

Discussion
You specify the source string's encoding in the Unicode mapping structure that you pass to the function `CreateTextToUnicodeInfo` (page 1890) to obtain a Unicode converter object for the conversion. You pass the Unicode converter object returned by `CreateTextToUnicodeInfo` to `ConvertFromTextToUnicode` as the `iTextToUnicodeInfo` parameter.

In addition to converting a text string in any encoding to Unicode, the `ConvertFromTextToUnicode` function can map offsets for style or font information from the source text string to the returned converted string. The converter reads the application-supplied offsets, which apply to the source string, and returns the corresponding new offsets in the converted string. If you do not want the offsets at which font or style information occurs mapped to the resulting string, you should pass `NULL` for `iOffsetArray` and `0` (zero) for `iOffsetCount`.

Your application must allocate a buffer to hold the resulting converted string and pass a pointer to the buffer in the `oUnicodeStr` parameter. To determine the size of the output buffer to allocate, you should consider the size of the source string, its encoding type, and its content in relation to the resulting Unicode string.

For example, for 1-byte encodings, such as MacRoman, the Unicode string will be at least double the size (more if it uses noncomposed Unicode) for MacArabic and MacHebrew, the corresponding Unicode string could be up to six times as big. For most 2-byte encodings, for example Shift-JIS, the Unicode string will be less than double the size. For international robustness, your application should allocate a buffer three to four times larger than the source string. If the output Unicode text is actually UTF-8—which could occur beginning with the current release of the Text Encoding Conversion Manager, version 1.2.1—the UTF-8 buffer pointer must be cast to `UniCharArrayPtr` before it can be passed as the `oUnicodeStr` parameter. Also, the output buffer length will have a wider range of variation than for UTF-16; for ASCII input, the output will be the same size; for Han input, the output will be twice as big, and so on.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

ConvertFromUnicodeToPString

Converts a Unicode string to Pascal in a Mac OS text encoding.

```
OSStatus ConvertFromUnicodeToPString (
   UnicodeToTextInfo iUnicodeToTextInfo,
   ByteCount iUnicodeLen,
   const UniChar iUnicodeStr[],
   Str255 oPascalStr
);
```

Parameters

iUnicodeToTextInfo

A Unicode converter object. You use the `CreateUnicodeToTextInfo` or `CreateUnicodeToTextInfoByEncoding` function to obtain the Unicode converter object for the conversion.

iUnicodeLen

The length in bytes of the Unicode string to be converted. This is the string your application provides in the `iUnicodeStr` parameter.

iUnicodeStr

A pointer to an array containing the Unicode string to be converted.

oPascalStr

A buffer. On return, the converted Pascal string returned by the function.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Discussion

The `ConvertFromUnicodeToPString` function provides an easy and efficient way to convert a Unicode string to a Pascal string in a Mac OS text encoding without incurring the overhead associated with use of the function `ConvertFromUnicodeToText` (page 1883) or `ConvertFromUnicodeToScriptCodeRun` (page 1880).

If necessary, this function uses the loose mapping and fallback characters to map the text elements of the string. For fallback mappings, it uses the handler associated with the Unicode converter object.

Availability

Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`UnicodeConverter.h`

ConvertFromUnicodeToScriptCodeRun

Converts a string from Unicode to one or more scripts.

```
OSStatus ConvertFromUnicodeToScriptCodeRun (
   UnicodeToTextRunInfo iUnicodeToTextInfo,
   ByteCount iUnicodeLen,
   const UniChar iUnicodeStr[],
   OptionBits iControlFlags,
   ItemCount iOffsetCount,
   const ByteOffset iOffsetArray[],
   ItemCount *oOffsetCount,
   ByteOffset oOffsetArray[],
   ByteCount iOutputBufLen,
   ByteCount *oInputRead,
   ByteCount *oOutputLen,
   LogicalAddress oOutputStr,
   ItemCount iScriptRunBufLen,
   ItemCount *oScriptRunOutLen,
   ScriptCodeRun oScriptCodeRuns[]
);
```

Parameters

iUnicodeToTextInfo

You use the function `CreateUnicodeToTextRunInfoByScriptCode` (page 1896) to obtain a Unicode converter object to specify for this parameter.

iUnicodeLen

The length in bytes of the Unicode string to be converted.

iUnicodeStr

A pointer to the Unicode string to be converted.

iControlFlags

Conversion control flags. The following constants define the masks for control flags valid for this parameter. You can use "Conversion Masks" (page 1972) and "Directionality Masks" (page 1976) to set the `iControlFlags` parameter.

If the text-run control flag is clear, `ConvertFromUnicodeToScriptCodeRun` attempts to convert the Unicode text to the single script from the list of scripts in the Unicode converter object that produces the best result, that is, that provides for the greatest amount of source text conversion. If the complete source text can be converted into more than one of the scripts specified in the array, then the converter chooses among them based on their order in the array. If this flag is clear, the `oScriptCodeRuns` parameter always points to a value equal to 1.

If you set the use-fallbacks control flag, the converter uses the default fallback characters for the current script. If the converter cannot handle a character using the current encoding, even using fallbacks, the converter attempts to convert the character using the other scripts, beginning with the first one specified in the list and skipping the one where it failed.

If you set the `kUnicodeTextRunBit` control flag, the converter attempts to convert the complete Unicode text string into the first script specified in the Unicode mapping structures array you passed to `CreateUnicodeToTextRunInfo`, `CreateUnicodeToTextRunInfoByEncoding`, or `CreateUnicodeToTextRunInfoByScriptCode` to create the Unicode converter object used for this conversion. If it cannot do this, the converter then attempts to convert the first text element that failed to the remaining scripts, in their specified order in the array. What the converter does with the next text element depends on the setting of the keep-same-encoding control flag:

If the keep-same-encoding control flag is clear, the converter returns to the original script and attempts to continue conversion with that script; this is equivalent to converting each text element to the first one that works, in the order specified.

If the Unicode-keep-same-encoding control flag is set, the converter continues with the new destination script until it encounters a text element that cannot be converted using the new script. This attempts to minimize the number of script code changes in the output text. When the converter cannot convert a text element using any of the scripts in the list and the Unicode-keep-same-encoding control flag is set, the converter uses the fallbacks default characters for the current script.

iOffsetCount

The number of offsets in the array pointed to by the `iOffsetArray` parameter. Your application supplies this value. The number of entries in `iOffsetArray` must be fewer than half the number of bytes specified in `iUnicodeLen`. If you don't want offsets returned to you, specify `0` (zero) for this parameter.

iOffsetArray

An array of type `ByteOffset`. On input, you specify the array that contains an ordered list of significant byte offsets pertaining to the source Unicode string. These offsets may identify font or style changes, for example, in the Unicode string. If you don't want offsets returned to your application, specify `NULL` for this parameter and `0` (zero) for `iOffsetCount`.

oOffsetCount

On return, a pointer to the number of offsets that were mapped in the output stream.

oOffsetArray

An array of type `ByteOffset`. On return, this array contains the corresponding new offsets for the resulting converted string.

iOutputBufLen

The length in bytes of the output buffer pointed to by the `oOutputStr` parameter. Your application supplies this buffer to hold the returned converted string. The `oOutputLen` parameter may return a byte count that is less than this value if the converted byte string is smaller than the buffer size you allocated.

oInputRead

On return, a pointer to the number of bytes of the Unicode source string that were converted. If the function returns a result code other than `noErr`, then this parameter returns the number of bytes that were converted before the error occurred.

oOutputLen

On return, a pointer to the length in bytes of the converted string.

oOutputStr

A buffer address. On input, this value points to the beginning of the buffer for the converted string. On return, this buffer contains the converted string in one or more encodings. When an error occurs, the `ConvertFromUnicodeToScriptCodeRun` function returns the converted string up to the character that caused the error.

iScriptRunBufLen

The number of script code run elements you allocated for the script code run array pointed to by the `oScriptCodeRuns` parameter. The converter returns the number of valid script code runs in the location pointed to by `oScriptRunOutLen`. Each entry in the script code run array specifies the beginning offset in the converted text and its associated script code.

oScriptRunOutLen

A pointer to a value of type `ItemCount`. On output, this value contains the number of valid script code runs returned in the `oScriptCodeRuns` parameter.

oScriptCodeRuns

An array of elements of type `ScriptCodeRun`. Your application should allocate an array with the number of elements you specify in the `iScriptRunBufLen` parameter. On return, this array contains the script code runs for the converted text string. Each entry in the array specifies the beginning offset in the converted text string and the associated script code specification.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
To use the `ConvertFromUnicodeToScriptCodeRun` function, you must first set up an array of script codes containing in order of precedence the scripts to be used for the conversion. To create a Unicode converter object, you call the function `CreateUnicodeToTextRunInfoByScriptCode` (page 1896). You pass the returned Unicode converter object as the `iUnicodeToTextInfo` parameter when you call the `ConvertFromUnicodeToScriptCodeRun` function.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

ConvertFromUnicodeToText

Converts a Unicode text string to the destination encoding you specify.

```
OSStatus ConvertFromUnicodeToText (
   UnicodeToTextInfo iUnicodeToTextInfo,
   ByteCount iUnicodeLen,
   const UniChar iUnicodeStr[],
   OptionBits iControlFlags,
   ItemCount iOffsetCount,
   const ByteOffset iOffsetArray[],
   ItemCount *oOffsetCount,
   ByteOffset oOffsetArray[],
   ByteCount iOutputBufLen,
   ByteCount *oInputRead,
   ByteCount *oOutputLen,
   LogicalAddress oOutputStr
);
```

Parameters

iUnicodeToTextInfo

A Unicode converter object of type `UnicodeToTextInfo` for converting text from Unicode. You use the function `CreateUnicodeToTextInfo` (page 1892) or `CreateUnicodeToTextInfoByEncoding` (page 1893) to obtain a Unicode converter object to specify for this parameter. This function modifies the contents of the `iUnicodeToTextInfo` parameter.

iUnicodeLen

The length in bytes of the Unicode string to be converted.

iUnicodeStr

A pointer to the Unicode string to be converted. If the input text is UTF-8, which is supported for versions 1.2.1 or later of the converter, you must cast the UTF-8 buffer pointer to `ConstUniCharArrayPtr` before you can pass it as this parameter.

iControlFlags

Conversion control flags. You can use "Conversion Masks" (page 1972) and "Directionality Masks" (page 1976) to set the `iControlFlags` parameter.

iOffsetCount

The number of offsets contained in the array provided by the `iOffsetArray` parameter. Your application supplies this value. If you don't want offsets returned to you, specify `0` (zero) for this parameter.

iOffsetArray

An array of type `ByteOffset`. On input, you specify the array that gives an ordered list of significant byte offsets pertaining to the Unicode source string to be converted. These offsets may identify font or style changes, for example, in the source string. If you don't want offsets returned to your application, specify `NULL` for this parameter and `0` (zero) for `iOffsetCount`. All offsets must be less than `iUnicodeLen`.

oOffsetCount

On return, a pointer to the number of offsets that were mapped in the output stream.

oOffsetArray

An array of type `ByteOffset`. On return, this array contains the corresponding new offsets for the converted string in the new encoding.

iOutputBufLen

The length in bytes of the output buffer pointed to by the `oOutputStr` parameter. Your application supplies this buffer to hold the returned converted string. The `oOutputLen` parameter may return a byte count that is less than this value if the converted byte string is smaller than the buffer size you allocated.

oInputRead

On return, a pointer to a the number of bytes of the Unicode string that were converted. If the function returns a `kTECUnmappableElementErr` result code, this parameter returns the number of bytes that were converted before the error occurred.

oOutputLen

On return, a pointer to the length in bytes of the converted text stream.

oOutputStr

A value of type `LogicalAddress`. On input, this value points to a buffer for the converted string. On return, the buffer holds the converted text string. (For guidelines on estimating the size of the buffer needed, see the following discussion.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
This function can also map offsets for style or font information from the source text string to the returned converted string. The converter reads the application-supplied offsets and returns the corresponding new offsets in the converted string. If you do not want font or style information offsets mapped to the resulting string, you should pass `NULL` for `iOffsetArray` and `0` (zero) for `iOffsetCount`.

Your application must allocate a buffer to hold the resulting converted string and pass a pointer to the buffer in the `oOutputStr` parameter. To determine the size of the output buffer to allocate, you should consider the size and content of the Unicode source string in relation to the type of encoding to which it will be converted. For example, for many encodings, such as MacRoman and Shift-JIS, the size of the returned string will be between half the size and the same size as the source Unicode string. However, for some encodings that are not Mac OS ones, such as EUC-JP, which has some 3-byte characters for Kanji, the returned string could be larger than the source Unicode string. For MacArabic and MacHebrew, the result will usually be less than half the size of the Unicode string.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

ConvertFromUnicodeToTextRun

Converts a string from Unicode to one or more encodings.

```
OSStatus ConvertFromUnicodeToTextRun (
   UnicodeToTextRunInfo iUnicodeToTextInfo,
   ByteCount iUnicodeLen,
   const UniChar iUnicodeStr[],
   OptionBits iControlFlags,
   ItemCount iOffsetCount,
   const ByteOffset iOffsetArray[],
   ItemCount *oOffsetCount,
   ByteOffset oOffsetArray[],
   ByteCount iOutputBufLen,
   ByteCount *oInputRead,
   ByteCount *oOutputLen,
   LogicalAddress oOutputStr,
   ItemCount iEncodingRunBufLen,
   ItemCount *oEncodingRunOutLen,
   TextEncodingRun oEncodingRuns[]
);
```

Parameters

iUnicodeToTextInfo

You use the function `CreateUnicodeToTextRunInfo` (page 1894), `CreateUnicodeToTextRunInfoByEncoding` (page 1895), or `CreateUnicodeToTextRunInfoByScriptCode` (page 1896) to obtain a Unicode converter object to specify for this parameter.

iUnicodeLen

The length in bytes of the Unicode string to be converted.

iUnicodeStr

A pointer to the Unicode string to be converted.

iControlFlags

Conversion control flags. The following constants define the masks for control flags valid for this parameter. You can use "Conversion Masks" (page 1972) and "Directionality Masks" (page 1976) to set the `iControlFlags` parameter.

If the text-run control flag is clear, `ConvertFromUnicodeToTextRun` attempts to convert the Unicode text to the single encoding it chooses from the list of encodings in the Unicode mapping structures array that you provide when you create the Unicode converter object. This is the encoding that produces the best result, that is, that provides for the greatest amount of source text conversion. If the complete source text can be converted into more than one of the encodings specified in the Unicode mapping structures array, then the converter chooses among them based on their order in the array. If this flag is clear, the `oEncodingRuns` parameter always points to a value equal to 1.

If you set the use-fallbacks control flag, the converter uses the default fallback characters for the current encoding. If the converter cannot handle a character using the current encoding, even using fallbacks, the converter attempts to convert the character using the other encodings, beginning with the first encoding specified in the list and skipping the encoding where it failed.

If you set the `kUnicodeTextRunBit` control flag, the converter attempts to convert the complete Unicode text string into the first encoding specified in the Unicode mapping structures array you passed to `CreateUnicodeToTextRunInfo`, `CreateUnicodeToTextRunInfoByEncoding`, or `CreateUnicodeToTextRunInfoByScriptCod`e when you created the Unicode converter object for this conversion. If it cannot do this, the converter then attempts to convert the first text element that failed to the remaining encodings, in their specified order in the array. What the converter does with the next text element depends on the setting of the keep-same-encoding control flag.

If the keep-same-encoding control flag is clear, the converter returns to the original encoding and attempts to continue conversion with that encoding; this is equivalent to converting each text element to the first encoding that works, in the order specified.

If the keep-same-encoding control flag is set, the converter continues with the new destination encoding until it encounters a text element that cannot be converted using the new encoding. This attempts to minimize the number of encoding changes in the output text. When the converter cannot convert a text element using any of the encodings in the list and the Unicode-keep-same-encoding control flag is set, the converter uses the fallbacks default characters for the current encoding.

iOffsetCount

The number of offsets in the array pointed to by the `iOffsetArray` parameter. Your application supplies this value. If you don't want offsets returned to you, specify `0` (zero) for this parameter.

iOffsetArray

An array of type `ByteOffset`. On input, you specify the array that contains an ordered list of significant byte offsets pertaining to the source Unicode string. These offsets may identify font or style changes, for example, in the Unicode string. If you don't want offsets returned to your application, specify `NULL` for this parameter and `0` (zero) for `iOffsetCount`. All offsets must be less than `iUnicodeLen`.

oOffsetCount

On return, a pointer to the number of offsets that were mapped in the output stream.

oOffsetArray

An array of type `ByteOffset`. On return, this array contains the corresponding new offsets for the resulting converted string.

iOutputBufLen

The length in bytes of the output buffer pointed to by the `oOutputStr` parameter. Your application supplies this buffer to hold the returned converted string. The `oOutputLen` parameter may return a byte count that is less than this value if the converted byte string is smaller than the buffer size you allocated.

oInputRead

On return, a pointer to the number of bytes of the Unicode source string that were converted. If the function returns a result code other than `noErr`, then this parameter returns the number of bytes that were converted before the error occurred.

oOutputLen

On return, a pointer to the length in bytes of the converted string.

oOutputStr

A value of type `LogicalAddress`. On input, this value points to the start of the buffer for the converted string. On output, this buffer contains the converted string in one or more encodings. When an error occurs, the `ConvertFromUnicodeToTextRun` function returns the converted string up to the character that caused the error. (For guidelines on estimating the size of the buffer needed, see the discussion following the parameter descriptions.

iEncodingRunBufLen

The number of text encoding run elements you allocated for the encoding run array pointed to by the `oEncodingRuns` parameter. The converter returns the number of valid encoding runs in the location pointed to by `oEncodingRunOutLen`. Each entry in the encoding runs array specifies the beginning offset in the converted text and its associated text encoding.

oEncodingRunOutLen

On return, a pointer to a the number of valid encoding runs returned in the `oEncodingRuns` parameter.

oEncodingRuns

On input, an array of structures of type `TextEncodingRun`. Your application should allocate an array with the number of elements you specify in the `iEncodingRunBufLen` parameter. On return, this array contains the encoding runs for the converted text string. Each entry in the encoding run array specifies the beginning offset in the converted text string and the associated encoding specification.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Discussion

To use the `ConvertFromUnicodeToTextRun` function, you must first set up an array of structures of type `UnicodeMapping` (page 1967) containing, in order of precedence, the mapping information for the conversion. To create a Unicode converter object, you call the `CreateUnicodeToTextRunInfo` function passing it the Unicode mapping array, or you can the `CreateUnicodeToTextRunInfoByEncoding` or `CreateUnicodeToTextRunInfoByScriptCode` functions, which take arrays of text encodings or script codes instead of an array of Unicode mappings. You pass the returned Unicode converter object as the `iUnicodeToTextInfo` parameter when you call the `ConvertFromUnicodeToTextRun` function.

Two of the control flags that you can set for the `iControlFlags` parameter allow you to control how the Unicode Converter uses the multiple encodings in converting the text string. These flags are explained in the description of the `iControlFlags` parameter. Here is a summary of how to use these two control flags:

- To keep the converted text in a single encoding, clear the text-run control flag.
- To keep as much contiguous converted text as possible in one encoding, set the text-run control flag and clear the keep-same-encoding control flag.
- To minimize the number of resulting encoding runs and the changes of destination encoding, set both the text-run and keep-same-encoding control flags.

The `ConvertFromUnicodeToTextRun` function returns the converted string in the array pointed to by the `oOutputStr` parameter. Beginning with the first text element in the `oOutputStr` array, the elements of the array pointed to by the `oEncodingRuns` parameter identify the encodings of the converted string. The

number of elements in the `oEncodingRuns` array may not correspond to the number of elements in the `oOutputStr` array. This is because the `oEncodingRuns` array includes only elements for the beginning of each new encoding run in the converted string.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

CountUnicodeMappings

Counts available mappings that meet the specified matching criteria.

```
OSStatus CountUnicodeMappings (
   OptionBits iFilter,
   ConstUnicodeMappingPtr iFindMapping,
   ItemCount *oActualCount
);
```

Parameters

iFilter

Filter control flags representing the six subfields of the Unicode mapping structure that this function uses to match against in determining which mappings on the system to return to your application. The filter control enumeration, described in "Unicode Matching Masks" (page 1980), define the constants for the subfield's flags and their masks. You can include in the search criteria any of the three text encoding subfields for both the Unicode encoding and the other specified encoding. For any flag not turned on, the subfield value is ignored and the function does not check the corresponding subfield of the mappings on the system.

iFindMapping

A structure of type `UnicodeMapping` (page 1967) containing the text encodings whose field values are to be matched.

oActualCount

On return, a pointer to the number of matching mappings found.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
You can filter on any of the three text encoding subfields of the Unicode mapping structure's `unicodeEncoding` specification and on any of the three text encoding subfields of the structure's `otherEncoding` specification. The `iFilter` parameter consists of a set of six control flags that you set to identify which of the corresponding six subfields to include in the match count. No filtering is performed on fields for which you do not set the corresponding filter control flag.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

CreateTextEncoding

Creates and returns a text encoding specification.

```
TextEncoding CreateTextEncoding (
   TextEncodingBase encodingBase,
   TextEncodingVariant encodingVariant,
   TextEncodingFormat encodingFormat
);
```

Parameters

encodingBase

A base text encoding.

encodingVariant

A variant of the base text encoding. To specify the default variant for the base encoding given in the `encodingBase` parameter, you can use the `kTextEncodingDefaultVariant` constant.

encodingFormat

A format for the base text encoding. To specify the default format for the base encoding, you can use the `kTextEncodingDefaultFormat` constant. If you want to obtain a `TextEncoding` value that references UTF-16 or UTF-8, pass `kUnicode16BitFormat` or `kUnicodeUTF8Format`.

Return Value

The text encoding specification that the function creates from the values you pass it.

Discussion

When you create a text encoding specification, the three values that you specify are packed into an unsigned integer, which you can then pass by value to the functions that use text encodings. See the data type `TextEncodingRun` (page 1965).

Availability

Available in CarbonLib 1.0 and later when Text Common 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Declared In

`TextCommon.h`

CreateTextToUnicodeInfo

Creates and returns a Unicode converter object containing information required for converting strings from a non-Unicode encoding to Unicode.

```
OSStatus CreateTextToUnicodeInfo (
   ConstUnicodeMappingPtr iUnicodeMapping,
   TextToUnicodeInfo *oTextToUnicodeInfo
);
```

Parameters

iUnicodeMapping

A pointer to a structure of type `UnicodeMapping`. Your application provides this structure to identify the mapping to use for the conversion. You must supply a value of type `TextEncoding` in the `unicodeEncoding` field of this structure. A `TextEncoding` is a triple composed of an encoding base, an encoding variant, and a format. You can obtain a `UnicodeMapping` (page 1967) value by calling the function `CreateTextEncoding`.

oTextToUnicodeInfo

On return, the Unicode converter object holds mapping table information you supplied as the `UnicodeMapping` parameter and state information related to the conversion. This information is required for conversion of a text stream in a non-Unicode encoding to Unicode.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Discussion

You pass a Unicode converter object returned from the function CreateTextToUnicodeInfo to the function `ConvertFromTextToUnicode` (page 1877) or `ConvertFromPStringToUnicode` (page 1876) to identify the information to be used for the conversion. These two functions modify the contents of the object.

You pass a Unicode converter object returned from `CreateTextToUnicodeInfo` to the function `TruncateForTextToUnicode` (page 1934) to identify the information to be used to truncate the string. This function does not modify the contents of the Unicode converter object.

If an error is returned, the Unicode converter object is invalid.

Availability

Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`UnicodeConverter.h`

CreateTextToUnicodeInfoByEncoding

Based on the given text encoding specification, creates and returns a Unicode converter object containing information required for converting strings from the specified non-Unicode encoding to Unicode.

```
OSStatus CreateTextToUnicodeInfoByEncoding (
   TextEncoding iEncoding,
   TextToUnicodeInfo *oTextToUnicodeInfo
);
```

Parameters

iEncoding

The text encoding specification for the source text.

oTextToUnicodeInfo

The Unicode converter object of type `TextToUnicodeInfo` (page 1966) returned by the function.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
You can use this function instead of the `CreateTextToUnicodeInfo` (page 1890) function when you do not need to create a Unicode mapping structure. You simply specify the text encoding of the source text. However, this method is less efficient because the text encoding parameter must be resolved internally into a Unicode mapping.

You cannot specify a version of Unicode. The function uses a 16-bit form of Unicode as the default.

You pass a Unicode converter object returned from `CreateTextToUnicodeInfoByEncoding` to the function `ConvertFromTextToUnicode` (page 1877) or `ConvertFromPStringToUnicode` (page 1876) to identify the information to be used for the conversion. These two functions modify the contents of the Unicode converter object.

You pass a Unicode converter object returned from `CreateTextToUnicodeInfoByEncoding` to the function `TruncateForTextToUnicode` (page 1934) to identify the information to be used to truncate the string. This function does not modify the contents of the Unicode converter object.

If you are converting the text stream to Unicode as an intermediary encoding, and then from Unicode to the final destination encoding, you use the function `CreateUnicodeToTextInfo` (page 1892) to create a Unicode converter object for the second part of the process.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Declared In
`UnicodeConverter.h`

CreateUnicodeToTextInfo

Creates and returns a Unicode converter object containing information required for converting strings from Unicode to a non-Unicode encoding.

```
OSStatus CreateUnicodeToTextInfo (
   ConstUnicodeMappingPtr iUnicodeMapping,
   UnicodeToTextInfo *oUnicodeToTextInfo
);
```

Parameters

iUnicodeMapping

A pointer to a structure of type `UnicodeMapping` (page 1967). Your application provides this structure to identify the mapping to be used for the conversion. The `unicodeEncoding` field of this structure can specify a Unicode format of `kUnicode16BitFormat` or `kUnicodeUTF8Format`. Note that the versions of the Unicode Converter prior to 1.2.1 do not support `kUnicodeUTF8Format`.

oUnicodeToTextInfo

On return, a pointer to a Unicode converter object that holds the mapping table information you supply as the `iUnicodeMapping` parameter and the state information related to the conversion. The information contained in the Unicode converter object is required for the conversion of a Unicode string to a non-Unicode encoding.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
You pass the Unicode converter object returned from `CreateUnicodeToTextInfo` to the function `ConvertFromUnicodeToText` (page 1883) or `ConvertFromUnicodeToPString` (page 1879) to identify the information to be used for the conversion. These two functions modify the contents of the Unicode converter object.

If an error is returned, the Unicode converter object is invalid.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

CreateUnicodeToTextInfoByEncoding

Based on the given text encoding specification for the converted text, creates and returns a Unicode converter object containing information required for converting strings from Unicode to the specified non-Unicode encoding.

```
OSStatus CreateUnicodeToTextInfoByEncoding (
   TextEncoding iEncoding,
   UnicodeToTextInfo *oUnicodeToTextInfo
);
```

Parameters

iEncoding

The text encoding specification for the destination, or converted, text.

oUnicodeToTextInfo

A pointer to a Unicode converter object of type `UnicodeToTextInfo` (page 1969).

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
You can use this function instead of the `CreateUnicodeToTextInfo` (page 1892) function to create a Unicode converter. However, this method is less efficient internally because the destination text encoding you specify must be resolved into a Unicode mapping. Using this function, you cannot specify a version of Unicode, so a default version of Unicode is used; 16-bit format is assumed.

You pass a Unicode converter object returned from the function `CreateUnicodeToTextInfoByEncoding` to the function `ConvertFromUnicodeToText` (page 1883) or `ConvertFromUnicodeToPString` (page 1879) to identify the information to be used for the conversion. These two functions modify the contents of the Unicode converter object.

You pass a Unicode converter object returned from `CreateUnicodeToTextInfoByEncoding` to the function `TruncateForUnicodeToText` (page 1935) to identify the information to be used to truncate the string. This function does not modify the contents of the Unicode converter object.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

CreateUnicodeToTextRunInfo

Creates and returns a Unicode converter object containing the information required for converting a Unicode text string to strings in one or more non-Unicode encodings.

```
OSStatus CreateUnicodeToTextRunInfo (
   ItemCount iNumberOfMappings,
   const UnicodeMapping iUnicodeMappings[],
   UnicodeToTextRunInfo *oUnicodeToTextInfo
);
```

Parameters

iNumberOfMappings

The number of mappings specified by your application for converting from Unicode to any other encoding types, including other forms of Unicode. If you pass 0 for this parameter, the converter will use all of the scripts installed in the system. The primary script is the one with highest priority; `ScriptOrder` (`'itlm'` resource) determines the priority of the rest. If you set the high-order bit for this parameter, the Unicode converter assumes that the `iEncodings` parameter contains a single element specifying the preferred encoding. This feature is supported for versions 1.2 or later of the converter.

iUnicodeMappings

A pointer to an array of structures of type `UnicodeMapping` (page 1967). Your application provides this structure to identify the mappings to be used for the conversion. The order in which you specify the mappings determines the priority of the destination encodings. For this function, the Unicode mapping structure can specify a Unicode format of `kUnicode16BitFormat` or `kUnicodeUTF8Format`. Note that the versions of the Unicode Converter prior to the Text Encoding Conversion Manager 1.2.1 do not support `kUnicodeUTF8Format`. Also, note that the `unicodeEncoding` field should be the same for all of the entries in `iUnicodeMappings`. If you pass `NULL` for the `iUnicodeMappings` parameter, the converter uses all of the scripts installed in the system, assuming the default version of Unicode with 16-bit format. The primary script is the one with the highest priority and `ScriptOrder`(`'itlm'` resource) determines the priority of the rest. This is supported beginning with version 1.2 of the Text Encoding Conversion Manager.

oUnicodeToTextInfo

A pointer to a Unicode converter object for converting Unicode text strings to strings in one or more non-Unicode encodings. On return, a pointer to a Unicode converter object that holds the mapping table information you supply as the `iUnicodeMappings` parameter and the state information related to the conversion.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
You pass a Unicode converter object returned from the function `CreateUnicodeToTextRunInfo` to the function `ConvertFromUnicodeToTextRun` (page 1885) or `ConvertFromUnicodeToScriptCodeRun` (page 1880) to identify the information to be used for the conversion. These two functions modify the contents of the Unicode converter object.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

CreateUnicodeToTextRunInfoByEncoding

Based on the given text encoding specifications for the converted text runs, creates and returns a Unicode converter object containing information required for converting strings from Unicode to one or more specified non-Unicode encodings.

```
OSStatus CreateUnicodeToTextRunInfoByEncoding (
   ItemCount iNumberOfEncodings,
   const TextEncoding iEncodings[],
   UnicodeToTextRunInfo *oUnicodeToTextInfo
);
```

Parameters

iNumberOfEncodings

The number of desired encodings. If you pass 0 for this parameter, the converter will use all of the scripts installed in the system. The primary script is the one with highest priority; `ScriptOrder`('itlm' resource) determines the priority of the rest. If you set the high-order bit for this parameter, the Unicode converter assumes that the `iEncodings` parameter contains a single element specifying the preferred encoding. This feature is supported for versions 1.2 or later of the converter.

iEncodings

An array of text encoding specifications for the desired encodings. Your application provides this structure to identify the encodings to be used for the conversion. The order in which you specify the encodings determines the priority of the destination encodings. If you pass `NULL` for this parameter, the converter will use all of the scripts installed in the system. The primary script is the one with highest priority and `ScriptOrder`('itlm' resource) determines the priority of the rest.This feature is supported for versions 1.2 or later of the converter.

oUnicodeToTextInfo

A pointer to a Unicode converter object for converting Unicode text strings to strings in one or more non-Unicode encodings. On return, a pointer to a Unicode converter object that holds the encodings you supply as the `iEncodings` parameter and the state information related to the conversion.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
You pass a Unicode converter object returned from `CreateUnicodeToTextRunInfoByEncoding` to the function `ConvertFromUnicodeToTextRun` (page 1885) or `ConvertFromUnicodeToScriptCodeRun` (page 1880) to identify the information to be used for the conversion. These two functions modify the contents of the Unicode converter object.

If an error is returned, the converter object is invalid.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

CreateUnicodeToTextRunInfoByScriptCode

Based on the given script codes for the converted text runs, creates and returns a Unicode converter object containing information required for converting strings from Unicode to one or more specified non-Unicode encodings.

```
OSStatus CreateUnicodeToTextRunInfoByScriptCode (
   ItemCount iNumberOfScriptCodes,
   const ScriptCode iScripts[],
   UnicodeToTextRunInfo *oUnicodeToTextInfo
);
```

Parameters

iNumberOfScriptCodes

The number of desired scripts. If you pass 0 for this parameter, the converter uses all the scripts installed in the system. In this case, the primary script is the one with highest priority; `ScriptOrder` (`'itlm'` resource) determines the priority of the rest. If you set the high-order bit for this parameter, the Unicode converter assumes that the `iScripts` parameter contains a single element specifying the preferred script. This feature is supported beginning with the Text Encoding Conversion Manager 1.2.

iScripts

An array of script codes for the desired scripts. Your application provides this structure to identify the scripts to be used for the conversion. The order in which you specify the scripts determines their priority. If you pass `NULL` for this parameter, the converter uses all of the scripts installed in the system. In this case, the primary script is the one with the highest priority and the priority order of the remaining scripts is defined by the `ScriptOrder(itlm resource)` resource. This feature is supported for versions 1.2 or later of the converter.

oUnicodeToTextInfo

A pointer to a Unicode converter object for converting Unicode text strings to strings in one or more non-Unicode encodings. On return, a pointer to Unicode converter object that holds the scripts you supply as the `iScripts` parameter and the state information related to the conversion.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
You pass a Unicode converter object returned from `CreateUnicodeToTextRunInfoByScriptCode` to the function `ConvertFromUnicodeToTextRun` (page 1885) or `ConvertFromUnicodeToScriptCodeRun` (page 1880) to identify the information to be used for the conversion. These two functions modify the contents of the Unicode converter object.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Declared In
`UnicodeConverter.h`

DisposeTextToUnicodeInfo

Releases the memory allocated for the specified Unicode converter object.

```
OSStatus DisposeTextToUnicodeInfo (
   TextToUnicodeInfo *ioTextToUnicodeInfo
);
```

Parameters

ioTextToUnicodeInfo

A pointer to a Unicode converter object of type `TextToUnicodeInfo` (page 1966), used for converting text to Unicode. On input, you specify the object to dispose. It must be an object which your application created using the function `CreateTextToUnicodeInfo` (page 1890) or `CreateTextToUnicodeInfoByEncoding` (page 1891). You must not point to any other type of Unicode converter object. Your application should not use this function with the same structure more than once.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026). If your application specifies an invalid Unicode converter object, such as `NULL`, the function returns a `paramErr` result code.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

DisposeUnicodeToTextFallbackUPP

Disposes of a a new universal procedure pointer (UPP) to a Unicode-to-text fallback callback.

```
void DisposeUnicodeToTextFallbackUPP (
   UnicodeToTextFallbackUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion
See the callback `UnicodeToTextFallbackProcPtr` (page 1953) for more information.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

DisposeUnicodeToTextInfo

Releases the memory allocated for the specified Unicode converter object.

```
OSStatus DisposeUnicodeToTextInfo (
   UnicodeToTextInfo *ioUnicodeToTextInfo
);
```

Parameters

ioUnicodeToTextInfo

A pointer to a Unicode converter object for converting from Unicode to a non-Unicode encoding. You specify a Unicode converter object that your application created using the function `CreateUnicodeToTextInfo` (page 1892) or `CreateUnicodeToTextInfoByEncoding` (page 1893). You must not point to any other type of Unicode converter object. Your application should not attempt to dispose of the same Unicode converter object more than once.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026). The function returns `noErr` if it disposes of the Unicode converter object successfully. If your application specifies an invalid Unicode converter object, such as `NULL`, the function returns a `paramErr` result code.

Availability

Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`UnicodeConverter.h`

DisposeUnicodeToTextRunInfo

Releases the memory allocated for the specified Unicode converter object.

```
OSStatus DisposeUnicodeToTextRunInfo (
   UnicodeToTextRunInfo *ioUnicodeToTextRunInfo
);
```

Parameters

ioUnicodeToTextRunInfo

A pointer to a Unicode converter object. On input, you specify a Unicode converter object that points to the conversion information to dispose. It must be an object which your application created using the function `CreateUnicodeToTextRunInfo` (page 1894), `CreateUnicodeToTextRunInfoByEncoding` (page 1895), or `CreateUnicodeToTextRunInfoByScriptCode` (page 1896). You must point to any other type of Unicode converter object. Your application should not use this function with the same structure more than once.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026). If your application specifies an invalid Unicode converter object, such as `NULL`, the function returns `paramErr`.

Availability

Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
UnicodeConverter.h

GetTextEncodingBase

Returns the base encoding of the specified text encoding.

```
TextEncodingBase GetTextEncodingBase (
   TextEncoding encoding
);
```

Parameters

encoding
A text encoding specification whose base encoding you want to obtain.

Return Value
The base encoding portion of the specified text encoding.

Discussion
See the data type TextEncodingRun (page 1965)

Availability
Available in CarbonLib 1.0 and later when Text Common 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
TextCommon.h

GetTextEncodingFormat

Returns the format value of the specified text encoding.

```
TextEncodingFormat GetTextEncodingFormat (
   TextEncoding encoding
);
```

Parameters

encoding
A text encoding specification.

Return Value
The text encoding format value contained in the text encoding you specified.

Availability
Available in CarbonLib 1.0 and later when Text Common 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
TextCommon.h

GetTextEncodingName

Returns the localized name for a specified text encoding.

```
OSStatus GetTextEncodingName (
   TextEncoding iEncoding,
   TextEncodingNameSelector iNamePartSelector,
   RegionCode iPreferredRegion,
   TextEncoding iPreferredEncoding,
   ByteCount iOutputBufLen,
   ByteCount *oNameLength,
   RegionCode *oActualRegion,
   TextEncoding *oActualEncoding,
   TextPtr oEncodingName
);
```

Parameters

iEncoding

A text encoding specification whose name you want to obtain.

iNamePartSelector

The portion of the encoding name you want to obtain. See "Text Encoding Name Selectors" (page 2012) for a list of possible values.

iPreferredRegion

The preferred region to use for the name. You can specify a Mac OS region code (which also implies a language) for this parameter. If the function cannot return the name for the preferred region, it returns the name using a region code with the same language or in a default language (for example, English).

iPreferredEncoding

The preferred encoding to use for the name. For example, ASCII, Mac OS Roman, or Shift-JIS. If the function cannot return the name using the preferred encoding, it returns the name using another encoding, such as Unicode or ASCII.

iOutputBufLen

The length in bytes of the output buffer that your application provides for the returned encoding name.

oNameLength

A pointer to a value of type `ByteCount`. On return, this parameter holds the actual length, in bytes, of the text encoding name. The value represents the full length of the name, which might be greater than the size of the output buffer, specified by the `iOutputBufLen` parameter. The length of the portion of the name actually contained in the output buffer is the smaller of `oNameLength` and `iOutputBufLen`.

oActualRegion

A pointer to a value of type `RegionCode`. On return, this parameter holds the actual region associated with the returned encoding name.

oActualEncoding

A pointer to a value of type `TextEncoding`. On return, this parameter holds the actual encoding associated with the returned encoding name.

oEncodingName

A pointer to a buffer you provide. On return, this parameter holds the text encoding name.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
Names returned by `GetTextEncodingName` (in the buffer referred to by `oEncodingName`) can contain parentheses and other menu item meta characters, and so cannot be used with `AppendMenu` or `InsertMenuItem`. You can use them with `SetMenuItemText`.

This function can return resources and memory errors, and the following result codes:

- `kTextUnsupportedEncodingErr`, which indicates that the encoding whose name you want to obtain is not supported.
- `kTECMissingTableErr`, which indicates the name resource associated with the encoding is missing.
- `kTECTableFormatErr` or `kTECTableCheckSumErr`, which indicates that the name resource associated with that encoding is invalid.

Availability
Available in CarbonLib 1.0 and later when Text Common 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextCommon.h`

GetTextEncodingVariant

Returns the variant from the specified text encoding.

```
TextEncodingVariant GetTextEncodingVariant (
   TextEncoding encoding
);
```

Parameters
encoding
A text encoding specification.

Return Value
The text encoding variant portion of the specified text encoding.

Availability
Available in CarbonLib 1.0 and later when Text Common 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Declared In
`TextCommon.h`

InvokeUnicodeToTextFallbackUPP

Calls your Unicode-to-text fallback callback.

```
OSStatus InvokeUnicodeToTextFallbackUPP (
   UniChar *iSrcUniStr,
   ByteCount iSrcUniStrLen,
   ByteCount *oSrcConvLen,
   TextPtr oDestStr,
   ByteCount iDestStrLen,
   ByteCount *oDestConvLen,
   LogicalAddress iInfoPtr,
   ConstUnicodeMappingPtr iUnicodeMappingPtr,
   UnicodeToTextFallbackUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeUnicodeToTextFallbackUPP`, as the system calls your Unicode-to-text fallback callback for you. See the callback `UnicodeToTextFallbackProcPtr` (page 1953) for more information.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

NearestMacTextEncodings

Obtains the best and alternate Mac text encoding.

```
OSStatus NearestMacTextEncodings (
   TextEncoding generalEncoding,
   TextEncoding *bestMacEncoding,
   TextEncoding *alternateMacEncoding
);
```

Parameters

`generalEncoding`
The text encoding for which you want to obtain a Mac text encoding.

`bestMacEncoding`
On return, the Mac text encoding that best matches the encoding specified by the `generalEncoding` parameter.

`alternateMacEncoding`
On return, the Mac text encoding that is the second best match for the encoding specified by the `generalEncoding` parameter.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Availability
Available in CarbonLib 1.0 and later when Text Common 1.5 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextCommon.h`

NewUnicodeToTextFallbackUPP

Creates a new universal procedure pointer (UPP) to a Unicode-to-text fallback callback.

```
UnicodeToTextFallbackUPP NewUnicodeToTextFallbackUPP (
   UnicodeToTextFallbackProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your Unicode-to-text fallback callback.

Return Value

On return, a UPP to the Unicode-to-text fallback callback.

Discussion

See the callback `UnicodeToTextFallbackProcPtr` (page 1953) for more information.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Declared In

`UnicodeConverter.h`

QueryUnicodeMappings

Returns a list of the conversion mappings available on the system that meet specified matching criteria and returns the number of mappings found.

```
OSStatus QueryUnicodeMappings (
   OptionBits iFilter,
   ConstUnicodeMappingPtr iFindMapping,
   ItemCount iMaxCount,
   ItemCount *oActualCount,
   UnicodeMapping oReturnedMappings[]
);
```

Parameters

iFilter

Filter control flags representing the six values given in the Unicode mapping structure that this function uses to match against in determining which mappings on the system to return to your application. The filter control flag enumerations, described in "Unicode Matching Masks" (page 1980), define the constants for the flags and their masks. You can include in the search criteria any of the three text encoding values—base, variant, and format—for both the Unicode encoding and the other specified encoding. For any flag not turned on, the value is ignored the function does not check the corresponding value of the mapping tables on the system.

iFindMapping

A structure of type `UnicodeMapping` (page 1967) containing the text encodings whose values are to be matched.

iMaxCount

The maximum number of mappings that can be returned. You provide this value to identify the number of elements in the array pointed to by the `oReturnedMappings` parameter that your application allocated. If the function identifies more matching mappings than the array can hold, it returns as many of them as fit. The function also returns a `kTECArrayFullErr` in this case.

oActualCount

On return, a pointer to the number of matching mappings found. This number may be greater than the number of mappings specified by `iMaxCount` if more matching mappings are found than can fit in the `oReturnedMappings` array.

oReturnedMappings

A pointer to an array of structures of type `UnicodeMapping` (page 1967). On input, this pointer refers to an array for the matching mappings returned by the function. To allocate sufficient elements for the array, you can use the function `CountUnicodeMappings` (page 1889) to determine the number of mappings returned for given values of the `iFilter` and `iFindMapping` parameters. On return, a pointer to an array that holds the matching mappings. If there are more matches than the array can hold, the function returns as many of them as will fit and a `kTECBufferBelowMinimumSizeErr` error result. The `oActualCount` parameter identifies the number of matching mappings actually found, which may be greater than the number returned.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026). If the function returns a `noErr` result code, the value retuned in the `oActualCount` parameter is less than or equal to the value returned in the `iMaxCount` parameter and the `oReturnedMappings` parameter contains all of the matching mappings found. If the function returns a `kTECArrayFullErr`, the function found more mappings than your `oReturnedMappings` array could accommodate.

Discussion

You can use the `QueryUnicodeMappings` function to obtain all mappings on the system up to the number allowed by your `oReturnedMappings` array by specifying a value of zero for the `iFilter` field.

You can use the function to obtain very specific mappings by setting individual filter control flags. You can filter on any of the three text encoding subfields of the Unicode mapping structure's `unicodeEncoding` specification and on any of the three text encoding subfields of the mapping's `otherEncoding` specification. The `iFilter` parameter consists of a set of six control flags that you set to identify which of the corresponding six subfields to include in the match. The list provided in the `oReturnedMappings` parameter will contain only mappings that match the fields of the Unicode mapping structure whose text encodings subfields you identify in the filter control flags. No filtering is performed on subfields for which you do not set the corresponding filter control flag.

For example, to obtain a list of all mappings in which one of the encodings is the default variant and default format of the Unicode 1.1 base encoding and the other encoding is the default variant and default format of a base encoding other than Unicode, you would set up the `iFilter` and `iFindMappings` parameter as follows. To set up these parameters, you use the constants defined for the text encoding bases, the text encoding default variants, the text encoding default formats, and the filter control flag bitmasks. In this example, the text encoding base field of the Unicode mapping structure's `otherEncoding` field is ignored, so you can specify any value for it. When you call `QueryUnicodeMappings`, passing it these parameters, the function will return a list of mappings between the Unicode encoding you specified and every other available encoding in which each non-Unicode base encoding shows up once because you specified its default variant and default format.

```
iFindMapping.unicodeMapping = CreateTextEncoding(
kTextEncodingUnicodeV1_1,
kTextEncodingDefaultVariant,
kTextEncodingDefaultFormat);
```

```
iFindMapping.otherEncoding = CreateTextEncoding(
kTextEncodingMacRoman,
kTextEncodingDefaultVariant,
kTextEncodingDefaultFormat);
iFilter = kUnicodeMatchUnicodeBaseMask |
kUnicodeMatchUnicodeVariantMask |
kUnicodeMatchUnicodeFormatMask |
kUnicodeMatchOtherVariantMask |
kUnicodeMatchOtherFormatMask;
```

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

ResetTextToUnicodeInfo

Reinitializes all state information kept by the context objects.

```
OSStatus ResetTextToUnicodeInfo (
   TextToUnicodeInfo ioTextToUnicodeInfo
);
```

Parameters
ioTextToUnicodeInfo

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.3 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

ResetUnicodeToTextInfo

Reinitializes all state information kept by a Unicode converter object.

```
OSStatus ResetUnicodeToTextInfo (
   UnicodeToTextInfo ioUnicodeToTextInfo
);
```

Parameters
ioUnicodeToTextInfo

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Declared In
`UnicodeConverter.h`

ResetUnicodeToTextRunInfo

Reinitializes all state information kept by the context objects in TextRun conversions.

```
OSStatus ResetUnicodeToTextRunInfo (
   UnicodeToTextRunInfo ioUnicodeToTextRunInfo
);
```

Parameters

ioUnicodeToTextRunInfo

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

ResolveDefaultTextEncoding

Returns a text encoding specification in which any meta-values have been resolved to real values. Currently, this affects only the base encoding values packed into the text encoding specification.

```
TextEncoding ResolveDefaultTextEncoding (
   TextEncoding encoding
);
```

Parameters

encoding

A text encoding specification possibly containing meta-values that you want to resolve to a text encoding specification containing only real values.

Return Value
A text encoding specification containing only real base encoding values.

Discussion
This function is useful for application developers who are providing APIs that take text encoding specifications as parameters. All APIs in the Unicode Converter and Text Encoding Converter perform this translation automatically.

Availability
Available in CarbonLib 1.0 and later when Text Common 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextCommon.h`

RevertTextEncodingToScriptInfo

Converts the given Mac OS text encoding specification to the corresponding script code and, if possible, language code and font name.

```
OSStatus RevertTextEncodingToScriptInfo (
   TextEncoding iEncoding,
   ScriptCode *oTextScriptID,
   LangCode *oTextLanguageID,
   Str255 oTextFontname
);
```

Parameters

iEncoding

The text encoding specification to be converted.

oTextScriptID

A pointer to a value of type `ScriptCode`. On return, a Mac OS script code that corresponds to the text encoding specification you identified in the `iEncoding` parameter. If you do not pass a pointer for this parameter, the function returns a `paramErr` result code.

oTextLanguageID

A pointer to a value of type `LangCode`. On input, if you do not want the function to return the language code, specify `NULL` as the value of this parameter. On return, the appropriate language code, if the language can be unambiguously derived from the text encoding specification, for example, Japanese, and you did not set the parameter to `NULL`.

If you do not specify `NULL` on input and the language is ambiguous—that is, the function cannot accurately derive it from the text encoding specification—the function returns a value of `kTextLanguageDontCare`.

oTextFontname

A Pascal string. On input, if you do not want the function to return the font name, specify `NULL` as the value of this parameter. On return, the name of the appropriate font if the font can be unambiguously derived from the text encoding specification, for example, Symbol, and you did not set the parameter to `NULL`.

If you do not specify `NULL` on input and the font is ambiguous—that is, the function cannot accurately derive it from the text encoding specification—the function returns a zero-length string.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026). The function returns `paramErr` if the text encoding specification input parameter value is invalid. The function returns a `kTECTableFormatErr` result code if the internal mapping tables used for translation are invalid. For a list of other possible result codes, see "Data Types".

Discussion

If you have applications that use Mac OS Script Manager and Font Manager functions, you can use the `RevertTextEncodingToScriptInfo` function to convert information in a text encoding specification into the appropriate Mac OS script code, language code, and font name, if they can be unambiguously derived. Your application can then use this information to display text to a user on the screen.

For more information see the `UpgradeScriptInfoToTextEncoding` (page 1937) function and "Base Text Encodings" (page 1982).

Availability

Available in CarbonLib 1.0 and later when Text Common 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
TextCommon.h

SetFallbackUnicodeToText

Specifies a fallback handler to be used for converting a Unicode text segment to another encoding when the Unicode Converter cannot convert the text using the mapping table specified by the Unicode converter object.

```
OSStatus SetFallbackUnicodeToText (
   UnicodeToTextInfo iUnicodeToTextInfo,
   UnicodeToTextFallbackUPP iFallback,
   OptionBits iControlFlags,
   LogicalAddress iInfoPtr
);
```

Parameters

iUnicodeToTextInfo

\The Unicode converter object to which the fallback handler is to be associated. You use the function CreateUnicodeToTextInfo (page 1892) or CreateUnicodeToTextInfoByEncoding (page 1893) to obtain a Unicode converter object of this type.

iFallback

A universal procedure pointer to the application-defined fallback routine. For a description of the function prototype that your fallback handler must adhere to and how to create your own fallback handler, see UnicodeToTextFallbackProcPtr (page 1953). You should use the NewUnicodeToTextFallbackProc macro to convert a pointer to your fallback handler into a UnicodeToTextFallbackUPP.

iControlFlags

Control flags that stipulate which fallback handler the Unicode Converter should call—the application-defined fallback handler or the default handler—if a fallback handler is required, and the sequence in which the Unicode Converter should call the fallback handlers if either can be used when the other fails or is unavailable. See "Fallback Handler Selectors" (page 1982).

iInfoPtr

A point to a block of memory to be passed to the application-defined fallback handler. The Unicode Converter passes this pointer to the application-defined fallback handler as the last parameter when it calls the fallback handler. Your application can use this memory block to store data required by your fallback handler whenever it is called. This is similar in use to a reference constant (refcon). If you don't need to use a memory block, specify NULL for this parameter.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
You can define multiple fallback handlers and associate them with different Unicode converter objects, depending on your requirements. See UnicodeToTextFallbackProcPtr (page 1953) for a description of how to create and install an application-defined fallback handler.

You can use a fallback handler when one of the Unicode conversion functions, ConvertFromUnicodeToText (page 1883) , ConvertFromUnicodeToTextRun (page 1885) , ConvertFromUnicodeToPString (page 1879) , and ConvertFromUnicodeToScriptCodeRun (page 1880) , cannot convert the text using the mapping table specified by the Unicode converter object passed to the function.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

SetFallbackUnicodeToTextRun

Specifies a fallback handler to be used for converting a Unicode text segment to another encoding when the Unicode Converter cannot convert the text using the mapping table specified by a Unicode converter object.

```
OSStatus SetFallbackUnicodeToTextRun (
   UnicodeToTextRunInfo iUnicodeToTextRunInfo,
   UnicodeToTextFallbackUPP iFallback,
   OptionBits iControlFlags,
   LogicalAddress iInfoPtr
);
```

Parameters

iUnicodeToTextRunInfo

The Unicode converter object to which the fallback handler is to be associated. You use the function `CreateUnicodeToTextRunInfo` (page 1894), `CreateUnicodeToTextRunInfoByEncoding` (page 1895), or `CreateUnicodeToTextRunInfoByScriptCode` (page 1896) to obtain a Unicode converter object to specify for this parameter.

iFallback

A universal procedure pointer to the application-defined fallback routine. For a description of the function prototype to which your fallback handler must adhere and how to create your own fallback handler, see `UnicodeToTextFallbackProcPtr` (page 1953). You should use the `NewUnicodeToTextFallbackProc` macro described in the discussion of the function `SetFallbackUnicodeToText` (page 1908).

iControlFlags

Control flags that stipulate which fallback handler the Unicode Converter should call—the application-defined fallback handler or the default handler—if a fallback handler is required, and the sequence in which the Unicode Converter should call the fallback handlers if either can be used when the other fails or is unavailable. See "Fallback Handler Selectors" (page 1982).

iInfoPtr

A pointer to a block of memory to be passed to the application-defined fallback handler. The Unicode Converter passes this pointer to the application-defined fallback handler as the last parameter when it calls the fallback handler. Your application can use this block to store data required by your fallback handler whenever it is called. This is similar in use to a reference constant (refcon). If you don't need to use a memory block, specify `NULL` for this parameter.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
You can define multiple fallback handlers and associate them with different Unicode converter objects, depending on your requirements. See `UnicodeToTextFallbackProcPtr` (page 1953) for a description of how to create and install an application-defined fallback handler.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

TECClearConverterContextInfo

Resets a converter object to its initial state so you can reuse it.

```
OSStatus TECClearConverterContextInfo (
   TECObjectRef encodingConverter
);
```

Parameters

encodingConverter

A reference to the text encoding converter object you want to reset. It can be a reference returned by the `TECCreateConverter` (page 1918), `TECCreateOneToManyConverter` (page 1920), or `TECCreateConverterFromPath` (page 1919) functions.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
It is more efficient to reuse an existing converter object than to create a new one that contains the same conversion information. This function clears the text string, but does not alter the source and destination encodings.

If you are converting multiple segments of a text string, you should not clear the converter object until you have converted all the text segments.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECClearSnifferContextInfo

Resets a sniffer object to its initial settings so you can reuse it.

```
OSStatus TECClearSnifferContextInfo (
   TECSnifferObjectRef encodingSniffer
);
```

Parameters

encodingSniffer

A pointer to the sniffer object you want to reset.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
Sniffers maintain state information about the input encoding buffer and the number of errors and features found for each encoding; this information allows a caller to progressively sniff an input buffer in sequential chunks. Before sniffing a buffer that contains completely new information you must clear any state information by calling `TECClearSnifferContextInfo`.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECConvertText

Converts a stream of text from a source encoding to a destination encoding. It uses the conversion path specified by the converter object you supply.

```
OSStatus TECConvertText (
   TECObjectRef encodingConverter,
   ConstTextPtr inputBuffer,
   ByteCount inputBufferLength,
   ByteCount *actualInputLength,
   TextPtr outputBuffer,
   ByteCount outputBufferLength,
   ByteCount *actualOutputLength
);
```

Parameters

encodingConverter
A reference to the text encoding converter object you want to use for the conversion. It can be a reference returned by the `TECCreateConverter` (page 1918) or `TECCreateConverterFromPath` (page 1919) functions.

inputBuffer
The stream of text you want to convert.

inputBufferLength
The length in bytes (`UInt8` or `unsigned char`) of the stream of text.

actualInputLength
On return, a pointer to the number of source text bytes that were converted from the input buffer.

outputBuffer
A pointer to a buffer for a byte stream. On output, the buffer holds the converted text.

outputBufferLength
The length in bytes of the `outputBuffer` parameter.

actualOutputLength
On return, a pointer to the number of bytes of converted text returned in the `outputBuffer` parameter.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026). If there is not enough memory available for `TECConvertText` to convert the text when allocating internal buffers, the function returns the appropriate Memory Manager result code.

Discussion
If the output buffer you allocate is too small to accommodate any of the converted text, the function fails. For best results, you should follow these guidelines when you allocate an output buffer:

- Base the buffer length on an estimate of the byte requirements of the destination encoding. Make sure you account for additional bytes needed by the destination encoding (for example, an escape sequence) in addition to the actual text.
- Always allocate a buffer at least 32 bytes long.
- If size is a concern, make sure the output buffer is at least large enough to hold a portion of the converted text. You can convert part of the text, then use the value of the `actualInputLength` parameter to identify the next byte to be taken and to determine how many bytes remain. To convert the remaining text, you simply call the function again with the remaining text and a new output buffer.
- If the destination encoding is a character encoding scheme—such as ISO-2022-JP, which begins in ASCII and switches to other coded character sets through limited combinations of escape sequences—then you need to allocate enough space to accommodate escape sequences that signal switches. ISO-2022-JP requires 3 to 5 bytes for an escape sequence preceding the 1-byte or 2-byte character it introduces. If you allocate a buffer that is less than 5 bytes, the TECConvertText function could fail, depending on the text being converted.

To make sure that you receive all of the converted text, you should call the function `TECFlushText` (page 1924) when you are finished converting all the text in a text stream.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECConvertTextToMultipleEncodings

Converts text in the source encoding to runs of text in multiple destination encodings. It uses the conversion path specified in the converter object you supply.

```
OSStatus TECConvertTextToMultipleEncodings (
   TECObjectRef encodingConverter,
   ConstTextPtr inputBuffer,
   ByteCount inputBufferLength,
   ByteCount *actualInputLength,
   TextPtr outputBuffer,
   ByteCount outputBufferLength,
   ByteCount *actualOutputLength,
   TextEncodingRun outEncodingsBuffer[],
   ItemCount maxOutEncodingRuns,
   ItemCount *actualOutEncodingRuns
);
```

Parameters

encodingConverter

The reference to the text encoding converter object to be used for the conversion. This is the reference returned by the function `TECCreateOneToManyConverter` (page 1920).

inputBuffer
The stream of text to be converted.

inputBufferLength
The length in bytes of the stream of text specified in the `inputBuffer` parameter.

actualInputLength
On return, a pointer to a the number of source text bytes that were converted.

outputBuffer
On return, a pointer to a buffer that holds the converted text.

outputBufferLength
The length in bytes of the `outputBuffer` parameter.

actualOutputLength
On return, a pointer to the number of bytes of the converted text returned in the `outputBuffer` parameter.

outEncodingsBuffer
An array of text encoding runs for output. Note that the actual byte size of this buffer should be `actualOutEncodingRuns * sizeof(TextEncodingRun)`.

maxOutEncodingRuns
The maximum number of runs that can fit in the `outEncodingsBuffer` array.

actualOutEncodingRuns
On return, a pointer to the number of runs in `outEncodingsBuffer` array.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026). If there is not enough memory available to convert the text when allocating internal buffers, the function returns the appropriate Memory Manager result code.

Discussion
For the function to return successfully, the output buffer you allocate must be large enough to accommodate the converted text. If the output buffer is too small to accommodate any converted text, the function will fail. For best results, you should follow these guidelines when you allocate an output buffer:

- Base the buffer length on an estimate of the byte requirements of the destination encoding. Make sure you account for additional bytes needed by the destination encoding (for example, an escape sequence) in addition to the actual text.
- Always allocate a buffer at least 32 bytes long.
- If size is a concern, make sure the output buffer is at least large enough to hold a portion of the converted text. You can convert part of the text, then use the value of the `actualInputLength` parameter to identify the next byte to be taken and to determine how many bytes remain. To convert the remaining text, you simply call the function again with the remaining text and a new output buffer.
- If the destination encoding is a character encoding scheme—such as ISO-2022-JP, which begins in ASCII and switches to other coded character sets through limited combinations of escape sequences—then you need to allocate enough space to accommodate escape sequences that signal switches. ISO-2022-JP requires 3 to 5 bytes for an escape sequence preceding the 1-byte or 2-byte character it introduces. If you allocate a buffer that is less than 5 bytes, the `TECConvertText` function could fail, depending on the text being converted.

The Text Encoding Converter creates internal buffers that hold intermediate results for indirect conversions

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECCountAvailableSniffers

Counts and returns the number of sniffers available in all installed plug-ins.

```
OSStatus TECCountAvailableSniffers (
   ItemCount *numberOfEncodings
);
```

Parameters

numberOfEncodings

On return, a pointer to the number of sniffers in all installed plug-ins. You can use this number to determine what size array to allocate for a parameter of the `TECGetAvailableSniffers` (page 1925) function.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
This function counts every instance of a sniffer. If different conversion plug-ins support a sniffer for the same encoding, the sniffer is counted more than once. Since the `TECGetAvailableSniffers` function ignores duplicate sniffers, `TECCountAvailableSniffers` may return a number greater than the number of array elements needed for the `availableSniffers[]` parameter of the `TECGetAvailableSniffers` function.

Availability
Supported in Carbon. Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECCountAvailableTextEncodings

Counts and returns the number of text encodings currently configured in the Text Encoding Converter.

```
OSStatus TECCountAvailableTextEncodings (
   ItemCount *numberEncodings
);
```

Parameters

numberEncodings

On return, a pointer to the number of currently supported text encodings. You use this value to determine the array size for a parameter of the `TECGetAvailableTextEncodings` (page 1926) function.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
The number of text encodings includes every instance of a text encoding. If different conversion plug-ins support the same text encoding, the text encoding will be counted more than once. For example, the Japanese Encodings plug-in supports Mac OS Japanese, and so does the Unicode Encodings plug-in. Since the `TECGetAvailableTextEncodings` function ignores duplicate text encoding specifications, `TECCountAvailableTextEncodings` may return a number greater than the number of array elements needed for the `availableEncodings []` parameter.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECCountDestinationTextEncodings

Counts and returns the number of destination encodings to which a specified source encoding can be converted in one step.

```
OSStatus TECCountDestinationTextEncodings (
   TextEncoding inputEncoding,
   ItemCount *numberOfEncodings
);
```

Parameters

`inputEncoding`
The text encoding specification describing the source text.

`numberOfEncodings`
On return, a pointer to the number of text encodings to which the source encoding can be converted in one step. You should use this to determine how large to make the array you pass to the `TECGetDestinationTextEncodings` (page 1926) function.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
This function counts every instance of an encoding. If different conversion plug-ins support the same direct text encoding, the direct text encoding is counted more than once.

Since the `TECGetDestinationTextEncodings` function ignores duplicate text encoding specifications, `TECCountDestinationTextEncodings` may return a number greater than the number of array elements needed for the `destinationEncodings[]` parameter.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Declared In
`TextEncodingConverter.h`

TECCountDirectTextEncodingConversions

Counts and returns the number of direct conversions currently configured in the Text Encoding Converter.

```
OSStatus TECCountDirectTextEncodingConversions (
   ItemCount *numberOfEncodings
);
```

Parameters

numberOfEncodings

On return, a pointer to the number of direct conversions. You should use this value to determine the array size for a parameter of the `TECGetDirectTextEncodingConversions` (page 1927) function.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Discussion

The number of direct conversions includes every instance of a conversion. If different conversion plug-ins support the same direct conversion, the direct conversion is counted more than once.

Since the `TECGetDirectTextEncodingConversions` (page 1927) function ignores duplicate direct conversions, `TECCountDirectTextEncodingConversions` may return a number greater than the number of array elements needed for the `directConversions` parameter.

Availability

Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`TextEncodingConverter.h`

TECCountMailTextEncodings

Counts and returns the number of currently supported e-mail encodings for a specified region.

```
OSStatus TECCountMailTextEncodings (
   RegionCode locale,
   ItemCount *numberEncodings
);
```

Parameters

locale

A Mac OS region code. A region code designates a combination of language, writing system, and geographic region; the region may not correspond to a particular country (for example, Swiss French or Arabic).

numberEncodings

On return, a pointer to the number of currently supported e-mail encodings for the region code. You use this number to determine what size array to allocate for a parameter of the `TECGetMailTextEncodings` (page 1929) function.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
This function counts every instance of an encoding. If different conversion plug-ins support the same direct text encoding, the direct text encoding is counted more than once. Since the `TECGetMailTextEncodings` function ignores duplicate text encoding specifications, `TECCountMailTextEncodings` may return a number greater than the number of array elements needed.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECCountSubTextEncodings

Counts and returns the number of subencodings a text encoding supports.

```
OSStatus TECCountSubTextEncodings (
   TextEncoding inputEncoding,
   ItemCount *numberOfEncodings
);
```

Parameters

inputEncoding
The text encoding specification that contains the subencodings.

numberOfEncodings
On return, a pointer to the number of currently supported subencodings. You use this value to determine the array size for a parameter of the `TECGetSubTextEncodings` (page 1930) function.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
Subencodings are text encodings that are embedded as part of a larger text encoding specification. For example, EUC-JP contains JIS Roman or ASCII, JIS X0208, JIS X0212, and half-width Katakana from JIS X0201. Not every encoding that can be broken into multiple encodings necessarily supports this routine. It's up to the plug-in developer to decide which encodings might be useful to break up. Subencodings are not the same as text encoding variants.

If an encoding can be converted to multiple runs of encodings (as indicated by a destination base encoding of `kTextEncodingMultiRun`), you can call the `TECGetSubTextEncodings` function to get the list of output encodings. See the `TECCreateOneToManyConverter` (page 1920) and `TECGetDestinationTextEncodings` (page 1926) functions for information.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECCountWebTextEncodings

Counts and returns the number of currently supported text encodings for a region code.

```
OSStatus TECCountWebTextEncodings (
   RegionCode locale,
   ItemCount *numberEncodings
);
```

Parameters

locale

A Mac OS region code indicating the locale for which you want to count encodings. A region code designates a combination of language, writing system, and geographic region; the region may not correspond to a particular country (for example, Swiss French or Arabic).

numberEncodings

On return, a pointer to the number of currently supported text encodings for a region code. You should use this number to determine how large to make the array you pass to the `TECGetWebTextEncodings` (page 1932) function.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Discussion

This function counts every instance of the same encoding. That is, if different conversion plug-ins support the same text encoding for a conversion process, the text encoding is counted more than once. Since the `TECGetWebTextEncodings` function ignores duplicate text encoding specifications, `TECCountWebTextEncodings` may return a number greater than the number of array elements needed for the `availableEncodings[]` parameter.

Availability

Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`TextEncodingConverter.h`

TECCreateConverter

Determines a conversion path for a source and destination encoding, then creates a text encoding converter object and returns a pointer to it.

```
OSStatus TECCreateConverter (
   TECObjectRef *newEncodingConverter,
   TextEncoding inputEncoding,
   TextEncoding outputEncoding
);
```

Parameters

newEncodingConverter

A pointer to a converter object. On return, this reference points to a newly created text converter object.

inputEncoding

The text encoding specification for the source text encoding.

outputEncoding
The text encoding specification for the destination text encoding.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
You use this converter object reference with conversion functions such as `TECConvertText` (page 1911) to convert text. This converter object describes the source, destination, and intermediate encodings; state information; and references to required plug-ins.

If the function does not find a direct conversion path, it creates an indirect conversion path. You can use the function `TECCreateConverterFromPath` (page 1919) to specify an explicit conversion path.

You must use the `TECDisposeConverter` (page 1921) function to remove a converter object.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECCreateConverterFromPath

Creates a converter object for a specific conversion path—from a source encoding through intermediate encodings to a destination encoding—and returns a pointer to it.

```
OSStatus TECCreateConverterFromPath (
   TECObjectRef *newEncodingConverter,
   const TextEncoding inPath[],
   ItemCount inEncodings
);
```

Parameters
newEncodingConverter
A pointer to a converter object reference. On return, the reference points to a newly created text converter object.

inPath
An ordered array of text encoding specifications, beginning with the source encoding specification and ending with the destination encoding specification. Each adjacent pair of text encodings must represent a conversion that is supported by the Text Encoding Converter.

inEncodings
The number of text encoding specifications in the `inPath` array.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
This function is faster than the function `TECCreateConverter` (page 1918) since it does not need to search for a conversion path. You can use the `TECGetDestinationTextEncodings` (page 1926) function to determine each step in the sequence from the source to the destination encoding.

To remove a converter object, you must call the function `TECDisposeConverter` (page 1921).

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECCreateOneToManyConverter

Determines a conversion path for the source encoding and destinations encodings you specify, creates a text encoding converter object, and returns a reference to it.

```
OSStatus TECCreateOneToManyConverter (
   TECObjectRef *newEncodingConverter,
   TextEncoding inputEncoding,
   ItemCount numOutputEncodings,
   const TextEncoding outputEncodings[]
);
```

Parameters

newEncodingConverter
A pointer to a converter object. On return, this points to a newly created one-to-many converter object.

inputEncoding
The text encoding specification for the source text encoding.

numOutputEncodings
The number of text encoding specifications in the `outputEncoding` array.

outputEncodings
An ordered array of text encoding specifications for the destination text encodings.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
You use this converter object reference with conversion functions such as `TECConvertTextToMultipleEncodings` (page 1912). The converter object describes the source, destination, and intermediate encodings; state information; and references to required plug-ins.

To remove a converter object, you must call the function `TECDisposeConverter` (page 1921).

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECCreateSniffer

Creates a sniffer object and returns a reference to it.

```
OSStatus TECCreateSniffer (
   TECSnifferObjectRef *encodingSniffer,
   TextEncoding testEncodings[],
   ItemCount numTextEncodings
);
```

Parameters

encodingSniffer

A pointer to a sniffer object reference, which is of type `TECSnifferObjectRef` (page 1964). On return, the reference pertains to the newly created sniffer object.

testEncodings

An array of text encoding specifications supplied by the caller; `TECCreateSniffer` creates a sniffer that can detect each of these encodings.

numTextEncodings

The number of text encoding specifications in the `testEncodings[]` array.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
The `TECCreateSniffer` function polls plug-ins for available sniffers, creates a sniffer object capable of sniffing each of the specified encodings that it can find a sniffer function for, and returns a reference to it. You use this sniffer object reference with sniffer functions such as `TECSniffTextEncoding` (page 1933). If no sniffer function is available for an encoding, no error is returned and `TECSniffTextEncoding` indicates later that the encoding was not examined.

To remove a sniffer object, you must call the function `TECDisposeSniffer` (page 1922).

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECDisposeConverter

Disposes of a converter object.

```
OSStatus TECDisposeConverter (
   TECObjectRef newEncodingConverter
);
```

Parameters

newEncodingConverter

A reference to the text encoding converter object you want to remove. This can be the reference returned by the `TECCreateConverter` (page 1918), `TECCreateConverterFromPath` (page 1919), or `TECCreateOneToManyConverter` (page 1920) functions.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
If you want to reuse the converter object for a different text stream with the same source and destination encoding, you should clear the converter object using the `TECClearConverterContextInfo` (page 1910) function rather than disposing of it and then creating a new converter object.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECDisposeSniffer

Disposes of a sniffer object.

```
OSStatus TECDisposeSniffer (
   TECSnifferObjectRef encodingSniffer
);
```

Parameters
encodingSniffer
The sniffer object reference you want to remove.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
This function releases all memory allocated to the sniffer object created by the `TECCreateSniffer` (page 1920) function.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECFlushMultipleEncodings

Flushes out any encodings that may be stored in a converter object's temporary buffers and shifts encodings back to their default state, if any.

```
OSStatus TECFlushMultipleEncodings (
   TECObjectRef encodingConverter,
   TextPtr outputBuffer,
   ByteCount outputBufferLength,
   ByteCount *actualOutputLength,
   TextEncodingRun outEncodingsBuffer[],
   ItemCount maxOutEncodingRuns,
   ItemCount *actualOutEncodingRuns
);
```

Parameters

encodingConverter

The reference to the text encoding converter object whose contents are to be flushed. This is the reference returned by the function `TECCreateOneToManyConverter` (page 1920).

outputBuffer

On return, a pointer to a buffer that holds the converted text. An error is returned is the buffer is not large enough to hold the entire converted text stream.

outputBufferLength

The length in bytes of the `outputBuffer` parameter.

actualOutputLength

On return, a pointer to a the actual number of bytes of the converted text returned in the `outputBuffer` parameter.

outEncodingsBuffer

An ordered array of text encoding runs for the destination text encoding. Note that the actual byte size of this buffer should be `actualOutEncodingRuns * sizeof(TextEncodingRun)`.

maxOutEncodingRuns

The maximum number of encoding runs that can fit in `outEncodingsBuffer[]`.

actualOutEncodingRuns

On return, a pointer to a the number of runs in the buffer during conversion.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Discussion

You should always call `TECFlushMultipleEncodings` at the end of the conversion process to flush out any data that may be stored in the temporary buffers of the text encoding converter object or to perform other end-of-encoding conversion tasks. Encodings such as ISO-2022-JP are reset to a default state when you use this function.

For the function to return successfully, the output buffer you allocate must be large enough to accommodate the converted text. If the output buffer is too small to accommodate any converted text, the function will fail. For best results, you should follow these guidelines when you allocate an output buffer:

- Base the buffer length on an estimate of the byte requirements of the destination encoding. Make sure you account for additional bytes needed by the destination encoding (for example, an escape sequence) in addition to the actual text.
- Always allocate a buffer at least 32 bytes long.
- If size is a concern, make sure the output buffer is at least large enough to hold a portion of the converted text. You can convert part of the text, then use the value of the `actualInputLength` parameter to identify the next byte to be taken and to determine how many bytes remain. To convert the remaining text, you simply call the function again with the remaining text and a new output buffer.

- If the destination encoding is a character encoding scheme—such as ISO-2022-JP, which begins in ASCII and switches to other coded character sets through limited combinations of escape sequences—then you need to allocate enough space to accommodate escape sequences that signal switches. ISO-2022-JP requires 3 to 5 bytes for an escape sequence preceding the 1-byte or 2-byte character it introduces. If you allocate a buffer that is less than 5 bytes, the `TECConvertText` function could fail, depending on the text being converted.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECFlushText

Flushes out any data in a converter object's temporary buffers and resets the converter object.

```
OSStatus TECFlushText (
   TECObjectRef encodingConverter,
   TextPtr outputBuffer,
   ByteCount outputBufferLength,
   ByteCount *actualOutputLength
);
```

Parameters

`encodingConverter`
A reference to the text converter object whose contents are to be flushed. This can be a reference returned by the `TECCreateConverter` (page 1918) or `TECCreateConverterFromPath` (page 1919) functions.

`outputBuffer`
On return, a pointer to a buffer that holds the converted text.

`outputBufferLength`
The length in bytes of the buffer provided by the `outputBuffer` parameter.

`actualOutputLength`
On return, a pointer to the number of bytes of converted text returned in the buffer specified by the `outputBuffer` parameter.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
You should always call `TECFlushText` when you finish converting a text stream. If you are converting a single stream in multiple chunks using multiple calls to `TECConvertText`, you only need to call `TECFlushText` after the last call to `TECConvertText` for that stream. The function uses the conversion path specified in the converter object you supply.

For the function to return successfully, the output buffer you allocate must be large enough to accommodate the flushed text. If the output buffer is too small to accommodate any flushed text, the function will fail. For best results, you should follow these guidelines when you allocate an output buffer:

- Base the buffer length on an estimate of the byte requirements of the destination encoding. Make sure you account for additional bytes needed by the destination encoding (for example, an escape sequence) in addition to the actual text.
- Always allocate a buffer at least 32 bytes long.

Encodings such as ISO-2022 that need to shift back to a certain default state at the end of a conversion can do so when this function is called.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECGetAvailableSniffers

Returns the list of sniffers available in all installed plug-ins.

```
OSStatus TECGetAvailableSniffers (
   TextEncoding availableSniffers[],
   ItemCount maxAvailableSniffers,
   ItemCount *actualAvailableSniffers
);
```

Parameters

availableSniffers
On return, an array of text encoding specifications that the available sniffers currently support. You should use the `TECCountAvailableSniffers` (page 1914) function to determine what size array to allocate.

maxAvailableSniffers
The number of text encoding specifications the `availableSniffers` array can contain.

actualAvailableSniffers
On return, a pointer to the number of text encodings in the `availableSniffers` array.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
This function ignores duplicate text encoding specifications. If you used the `TECCountAvailableSniffers` (page 1914) function to determine the size of the `TECGetAvailableSniffers` array, the number of available encodings may be fewer than the number of array elements, because `TECCountAvailableSniffers` includes duplicate text encoding specifications in its count.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECGetAvailableTextEncodings

Returns the text encoding specifications currently configured in the Text Encoding Converter.

```
OSStatus TECGetAvailableTextEncodings (
   TextEncoding availableEncodings[],
   ItemCount maxAvailableEncodings,
   ItemCount *actualAvailableEncodings
);
```

Parameters

availableEncodings

On return, an array of text encoding specifications. You should use the `TECCountAvailableTextEncodings` (page 1914) function to determine what size array to allocate.

maxAvailableEncodings

The number of text encoding specifications the `availableEncodings` array can contain.

actualAvailableEncodings

On return, a pointer to the number of text encodings returned in the `availableEncodings` array.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Discussion

This function ignores duplicate text encoding specifications. If you used the `TECCountAvailableTextEncodings` (page 1914) function to determine the size of the `availableEncodings` [] array, the number of encodings may be fewer than the number of array elements, because `TECCountAvailableTextEncodings` includes duplicate text encodings in its count.

Availability

Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`TextEncodingConverter.h`

TECGetDestinationTextEncodings

Returns the encoding specifications for all the destination text encodings to which the Text Encoding Converter can directly convert the specified source encoding.

```
OSStatus TECGetDestinationTextEncodings (
   TextEncoding inputEncoding,
   TextEncoding destinationEncodings[],
   ItemCount maxDestinationEncodings,
   ItemCount *actualDestinationEncodings
);
```

Parameters

inputEncoding

The text encoding specification describing the source text.

destinationEncodings

On return, an array of specifications for the destination encodings to which the converter can directly convert the source encoding. You should use the `TECCountDestinationTextEncodings` (page 1915) function to determine how large an array to allocate.

maxDestinationEncodings
The maximum number of destination text encodings that the array can contain.

actualDestinationEncodings
On return, a pointer to the number of text encoding specifications in the destination encodings array.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
This function ignores duplicate direct text encoding specifications. If you used the `TECCountDestinationTextEncodings` (page 1915) function to determine the size of the `destinationEncodings[]` array, the number of available encodings may be fewer than the number of array elements, because `TECCountDestinationTextEncodings` includes duplicates in its count.

You can display the names of these destination encodings to the user.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECGetDirectTextEncodingConversions

Returns the types of direct conversions currently configured in the Text Encoding Converter.

```
OSStatus TECGetDirectTextEncodingConversions (
   TECConversionInfo availableConversions[],
   ItemCount maxAvailableConversions,
   ItemCount *actualAvailableConversions
);
```

Parameters

availableConversions
An array composed of text encoding conversion information structures, each of which specifies a set of source and destination encodings for a type of conversion. See `TECConversionInfo` (page 1959) for more information. You should use the `TECGetDirectTextEncodingConversions` (page 1927) function to determine how large to make the array.

maxAvailableConversions
The maximum number of text encoding conversion information structures that the `directConversions` array can contain.

actualAvailableConversions
On return, a pointer to the number of text encoding conversion information structures returned in the `directConversions` array.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
This function ignores duplicate text encoding conversion information structures. If you used the `TECCountDirectTextEncodingConversions` (page 1916) function to determine the size of the `directConversions[]` array, the number of text encoding conversion information structures may be fewer than the number of array elements, because `TECCountDirectTextEncodingConversions` counts duplicate text encoding conversion information structures.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECGetEncodingList

Gets the list of destination encodings from a converter object.

```
OSStatus TECGetEncodingList (
   TECObjectRef encodingConverter,
   ItemCount *numEncodings,
   Handle *encodingList
);
```

Parameters

`encodingConverter`
A reference to the text encoding conversion object returned by the `TECCreateOneToManyConverter` (page 1920) function.

`numEncodings`
On return, a pointer to the number of encodings specified by the `encodingList` handle.

`encodingList`
A handle to an array of text encoding specifications. On return, it contains an array of text encoding specifications to which the converter object can convert. The memory for the array is allocated automatically by the Text Encoding Converter.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
The `TECDisposeConverter` function automatically disposes of the pointer for you. This means you should not reference the pointer after you have disposed of the converter object.

Plug-ins that perform one-to-many conversions use the `TECGetEncodingList` function to get the output encoding list from the converter object reference.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Declared In
`TextEncodingConverter.h`

TECGetInfo

Allocates a converter information structure of type `TECInfo` in the application heap using `NewHandle`, fills it out, and returns a handle.

```
OSStatus TECGetInfo (
   TECInfoHandle *tecInfo
);
```

Parameters

tecInfo

A handle to a structure of type `TECInfo` (page 1961) containing information about the converter.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026). This function can return memory errors.

Discussion

When you are finished with the handle, your application must dispose of it using `DisposeHandle`. You must also perform any required preflighting or memory rearrangement before calling `TECGetInfo`.

Availability

Available in CarbonLib 1.0 and later when Text Common 1.2.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`TextCommon.h`

TECGetMailTextEncodings

Returns the currently supported mail encoding specifications for a region code.

```
OSStatus TECGetMailTextEncodings (
   RegionCode locale,
   TextEncoding availableEncodings[],
   ItemCount maxAvailableEncodings,
   ItemCount *actualAvailableEncodings
);
```

Parameters

locale

A Mac OS region code. A region code designates a combination of language, writing system, and geographic region; the region may not correspond to a particular country (for example, Swiss French or Arabic).

availableEncodings

An array of text encoding specifications. On return, the array contains specifications for the e-mail text encodings for a region code. You should use the function `TECCountMailTextEncodings` (page 1916) function to determine what size array to allocate.

maxAvailableEncodings

The number of text encoding specifications the `availableEncodings` array can contain.

actualAvailableEncodings

On return, a pointer to the number of text encodings in the `availableEncodings` array.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
This function ignores duplicate text encoding specifications. If you used the `TECCountMailTextEncodings` (page 1916) function to determine the size of the `availableEncodings[]` array the number of available encodings may be fewer than the number of array elements, because `TECCountMailTextEncodings` includes duplicate text encoding specifications in its count.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECGetSubTextEncodings

Returns the text encoding specifications for the subencodings the encoding scheme supports.

```
OSStatus TECGetSubTextEncodings (
   TextEncoding inputEncoding,
   TextEncoding subEncodings[],
   ItemCount maxSubEncodings,
   ItemCount *actualSubEncodings
);
```

Parameters

inputEncoding
A text encoding specification.

subEncodings
On return, the array contains the specifications for the subencodings of the `inputEncoding` parameter. You should use the function `TECCountSubTextEncodings` (page 1917) function to determine what size an array to allocate.

maxSubEncodings
The number of text encoding specifications the `subEncodings` array can contain.

actualSubEncodings
On return, a pointer to number of subencodings in the `subEncodings` array.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
Subencodings are text encodings that are embedded as part of a larger text encoding specification. For example, EUC-JP contains JIS Roman or ASCII, JIS X0208, JIS X0212, and half-width Katakana from JIS X0201. Not every encoding that can be broken into multiple encodings necessarily supports this routine. It's up to the plug-in developer to decide which encodings might be useful to break up. Subencodings are not the same as text encoding variants

If an encoding can be converted to multiple runs of encodings (as indicated by a destination base encoding of `kTextEncodingMultiRun`), you can call the `TECGetSubTextEncodings` (page 1930) function to get the list of output encodings. See the `TECCreateOneToManyConverter` (page 1920) and `TECGetDestinationTextEncodings` (page 1926) functions for information about multiple output encoding run conversions.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECGetTextEncodingFromInternetName

Returns the Mac OS text encoding specification that corresponds to an Internet encoding name.

```
OSStatus TECGetTextEncodingFromInternetName (
   TextEncoding *textEncoding,
   ConstStr255Param encodingName
);
```

Parameters

textEncoding
On return, a pointer to a structure that contains a Mac OS text encoding specification.

encodingName
An Internet encoding name, in 7-bit US ASCII.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
Internet encoding names are stored as strings, while the Text Encoding Converter uses numeric values.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECGetTextEncodingInternetName

Returns the Internet encoding name that corresponds to a Mac OS text encoding.

```
OSStatus TECGetTextEncodingInternetName (
   TextEncoding textEncoding,
   Str255 encodingName
);
```

Parameters

textEncoding
A Mac OS text encoding specification.

encodingName
On return, the Internet encoding name, in 7-bit US ASCII. If there are several Internet encoding names for the same text encoding, the `encodingName` parameter contains the preferred name.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECGetWebTextEncodings

Returns the currently supported text encoding specifications for a region code.

```
OSStatus TECGetWebTextEncodings (
   RegionCode locale,
   TextEncoding availableEncodings[],
   ItemCount maxAvailableEncodings,
   ItemCount *actualAvailableEncodings
);
```

Parameters

locale
A Mac OS region code. A region code designates a combination of language, writing system, and geographic region and may not correspond to a particular country (for example, Swiss French or Arabic).

availableEncodings
On return, an array that contains specifications for the currently supported text encodings in the specified region. You should use the `TECCountWebTextEncodings` (page 1918) function to determine how large an array to allocate.

maxAvailableEncodings
The number of text encodings specifications the `availableEncodings` array can contain.

actualAvailableEncodings
On return, a pointer to the number of text encodings specifications in the `availableEncodings` array.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
This function ignores duplicate text encoding specifications. If you used the `TECCountWebTextEncodings` (page 1918) function to determine the size of the `availableEncodings[]` array the number of available encodings may be fewer than the number of array elements, because `TECCountWebTextEncodings` includes duplicate text encoding specifications in its count.

You can use the list of available encodings to create an encoding selection menu for a Web browser.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECSniffTextEncoding

Analyzes a text stream and returns the probable encodings in a ranked list, based on an array of possible encodings you supply. It also returns the number of errors and features for each encoding.

```
OSStatus TECSniffTextEncoding (
   TECSnifferObjectRef encodingSniffer,
   ConstTextPtr inputBuffer,
   ByteCount inputBufferLength,
   TextEncoding testEncodings[],
   ItemCount numTextEncodings,
   ItemCount numErrsArray[],
   ItemCount maxErrs,
   ItemCount numFeaturesArray[],
   ItemCount maxFeatures
);
```

Parameters

encodingSniffer
: A reference to a sniffer object.

inputBuffer
: The text to be sniffed.

inputBufferLength
: The length of the input buffer.

testEncodings
: An array of text encoding specifications. You must fill the array with the text encodings for which you want to sniff. On output, the array elements are reordered from the most likely to the least likely text encodings.

numTextEncodings
: The number of entries in the `testEncodings[]` parameter.

numErrsArray
: An array that must contain at least `numTextEncodings` elements. On return, an array of the number of errors found for each possible text encoding. The array elements are in the same order as the `testEncodings[]` array elements at output.

maxErrs
: The maximum number of errors a sniffer can encounter. The sniffer stops looking for an encoding after this number is reached.

numFeaturesArray
: An array of that must contain at least `numTextEncodings` elements. On return, an array of the number of features found for each possible text encoding. The array elements are in the same order as the `testEncodings[]` array elements at output.

maxFeatures
: The maximum number of features a sniffer can encounter. The sniffer stops looking for a features after this number is reached.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Discussion

An error indicates a code point or sequence that is illegal in the specified encoding. A feature indicates the presence of a sequence that is characteristic of that encoding.

For example, the byte sequence which is interpreted in Mac OS Roman as "é" could legally be interpreted either as Mac OS Roman text or as Mac OS Japanese text. Both sniffers would return zero errors, but the Mac OS Japanese sniffer would also return two features of Mac OS Japanese (representing two legal 2-byte characters.)

The arrays are returned in a ranked list with the most likely text encodings first. The results are sorted first by number of errors (fewest to most), then by number of features (most to fewest), and then by the original order in the list. On return, the most likely encoding is in `testEncodings[0]` or `testEncodings[1]`.

If an encoding is not examined, its number of errors and features are set to 0xFFFFFFFF, and the encoding is sorted to the end of the list.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Declared In
`TextEncodingConverter.h`

TruncateForTextToUnicode

Identifies where your application can safely break a multibyte string to be converted to Unicode so that the string is not broken in the middle of a multibyte character.

```
OSStatus TruncateForTextToUnicode (
   ConstTextToUnicodeInfo iTextToUnicodeInfo,
   ByteCount iSourceLen,
   ConstLogicalAddress iSourceStr,
   ByteCount iMaxLen,
   ByteCount *oTruncatedLen
);
```

Parameters

iTextToUnicodeInfo
The Unicode converter object of type `TextToUnicodeInfo` (page 1966) for the text string to be divided up with each segment properly truncated. The `TruncateForTextToUnicode` function does not modify the object's contents.

iSourceLen
The length in bytes of the multibyte string to be divided up.

iSourceStr
The address of the multibyte string to be divided up.

iMaxLen
The maximum allowable length of the string to be truncated. This must be less than or equal to `iSourceLen`.

oTruncatedLen
A pointer to a value of type `ByteCount`. On return, this value contains the length of the longest portion of the multibyte string, pointed to by `iSourceStr`, that is less than or equal to the length specified by `iMaxLen`. This identifies the byte after which you can break the string.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
Your application can use this function to break a string properly before you call the function `ConvertFromTextToUnicode` (page 1877) so that the string you pass it is terminated with complete characters. You can call this function repeatedly to properly divide up a text segment, each time identifying the new beginning of the string, until the last portion of the text is less than or equal to the maximum allowable length. Each time you use the function, you get a properly terminated string within the allowable length range.

Because the `TruncateForTextToUnicode` function does not modify the contents of the Unicode converter object, you can call this function safely between calls to the function `ConvertFromTextToUnicode` (page 1877).

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

TruncateForUnicodeToText

Identifies where your application can safely break a Unicode string to be converted to any encoding so that the string is broken in a way that preserves the text element integrity.

```
OSStatus TruncateForUnicodeToText (
   ConstUnicodeToTextInfo iUnicodeToTextInfo,
   ByteCount iSourceLen,
   const UniChar iSourceStr[],
   OptionBits iControlFlags,
   ByteCount iMaxLen,
   ByteCount *oTruncatedLen
);
```

Parameters

iUnicodeToTextInfo
A Unicode converter object `UnicodeToTextInfo` (page 1969) for the Unicode string to be divided up. The `TruncateForUnicodeToText` function does not modify the contents of this private structure.

iSourceLen
The length in bytes of the Unicode string to be divided up.

iSourceStr
A pointer to the Unicode string to be divided up.

iControlFlags
Truncation control flags. Specify the flag `kUnicodeStringUnterminatedMask` if truncating a buffer of text that belongs to a longer stream containing a subsequent buffer of text that could have characters belonging to a text element that begins at the end of the current buffer. If you set this flag, typically you would set the `iMaxLen` parameter equal to `iSourceLen`.

iMaxLen
The maximum allowable length of the string to be truncated. This must be less than or equal to `iSourceLen`.

oTruncatedLen

A pointer to a value of type `ByteCount`. On return, this value contains the length of the longest portion of the Unicode source string, pointed to by the `iSourceStr` parameter, that is less than or equal to the value of the `iMaxLen` parameter. This returned parameter identifies the byte after which you can truncate the string.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Discussion
Your application can use this function to divide up a Unicode string properly truncating each portion before you call `ConvertFromUnicodeToText` or `ConvertFromUnicodeToScriptCodeRun` to convert the string. You can call this function repeatedly to properly truncate a text segment, each time identifying the new beginning of the string, until the last portion of the text is less than or equal to the maximum allowable length. Each time you use the function, you get a properly terminated string within the allowable length range.

Because this function does not modify the contents of the Unicode converter object, you can call this function between conversion calls.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

UCGetCharProperty

Obtains the value associated with a property type for the specified `UniChar` characters.

```
OSStatus UCGetCharProperty (
   const UniChar *charPtr,
   UniCharCount textLength,
   UCCharPropertyType propType,
   UCCharPropertyValue *propValue
);
```

Parameters

charPtr

A pointer to the Unicode text whose property value you want to obtain.

textLength

The length of the text pointed to by `charPtr`.

propType

The property type for the `UniChar` character whose value you want to obtain. See "Unicode Character Property Types" (page 2020) for a list of the constants you can supply.

propValue

On return, the value associated with the property type specified by the `propType` parameter. See "Unicode Character Property Values" (page 2020) for a list of the constants that can be returned.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

Availability
Available in CarbonLib 1.0 and later when Text Common 1.5 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextCommon.h`

UpgradeScriptInfoToTextEncoding

Converts any combination of a Mac OS script code, a language code, a region code, and a font name to a text encoding.

```
OSStatus UpgradeScriptInfoToTextEncoding (
   ScriptCode iTextScriptID,
   LangCode iTextLanguageID,
   RegionCode iRegionID,
   ConstStr255Param iTextFontname,
   TextEncoding *oEncoding
);
```

Parameters

iTextScriptID
A valid Script Manager script code. The Mac OS Script Manager defines constants for script codes using this format: `sm`Xxx. To designate the system script, specify the meta-value of `smSystemScript`. To designate the current script based on the font specified in the graphics port (`grafPort`), specify the metavalue of `smCurrentScript`. To indicate that you do not want to provide a script code for this parameter, specify the constant `kTextScriptDontCare`.

iTextLanguageID
A valid Script Manager language code. The Mac OS Script Manager defines constants for language codes using this format: `lang`Xxx. To indicate that you do not want to provide a language code for this parameter, specify the constant `kTextLanguageDontCare`.

iRegionID
A valid Script Manager region code. The Mac OS Script Manager defines constants for region codes using this format: `ver`Xxx. To indicate that you do not want to provide a region code for this parameter, specify the constant `kTextRegionDontCare`.

iTextFontname
The name of a font associated with a particular text encoding specification, such as Symbol or Zapf Dingbats, or the name of any font that is currently installed on the system. To indicate that you do not want to provide a font name, specify a value of `NULL`.

oEncoding
A pointer to a value of type `TextEncoding`. On return, this value holds the text encoding specification that the function created from the other values you provided.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026). This function returns `paramErr` if two or more of the input parameter values conflict in some way—for example, the Mac OS language code does not belong to the script whose script code you specified, or if the input parameter values are invalid. The function returns a `kTECTableFormatErr` result code if the internal mapping tables used for translation are invalid.

Discussion
The `UpgradeScriptInfoToTextEncoding` function allows you to derive a text encoding specification from script codes, language codes, region codes, and font names. A one-to-one correspondence exists between many of the Script Manager's script codes and a particular Mac OS text encoding base value. However, because text encodings are a superset of script codes, some combinations of script code, language code, region code, and font name might result in a different text encoding base value than would be the case if the translation were based on the script code alone.

When you call the `UpgradeScriptInfoToTextEncoding` function, you can specify any combination of its parameters, but you must specify at least one.

If you don't specify an explicit value for a script, language, or region code parameter, you must pass the do-not-care constant appropriate to that parameter. If you do not specify an explicit value for `iTextFontName`, you must pass `NULL`. `UpgradeScriptInfoToTextEncoding` uses as much information as you supply to determine the equivalent text encoding or the closest approximation. If you provide more than one parameter, all parameters are checked against one another to ensure that they are valid in combination.

A font name, such as `'Symbol'` or `'Zapf Dingbats,'` can indicate a particular text encoding base. Other font names can indicate particular variants associated with a particular text encoding base. Otherwise, the font name is used to obtain a script code, and this script code will be checked against any script code you supply (in this case, the font must be installed; if it is not, the function returns a `paramErr` result code). If you do not supply either a language code or a region code and the script code you supply or the one that is derived matches the system script, then the system's localization is used to determine the appropriate region and language code. This is used for deriving text encoding base values that depend on region and language, such as `kTextEncodingMacTurkish`.

For more information see the `RevertTextEncodingToScriptInfo` (page 1907) function and "Base Text Encodings" (page 1982).

Availability
Available in CarbonLib 1.0 and later when Text Common 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextCommon.h`

Callbacks by Task

Setting Up a Fallback Handler

`UnicodeToTextFallbackProcPtr` (page 1953)
Defines a pointer to a function that converts a Unicode text element for which there is no destination encoding equivalent in the appropriate mapping table to the fallback character sequence defined by your fallback handler, and returns the converted character sequence to the Unicode Converter.

Setting Up a TEC Plug-in

`TECPluginGetPluginDispatchTablePtr` (page 1949)
Defines a pointer to a function that returns a pointer to a plug-in dispatch table.

`TECPluginNewEncodingConverterPtr` (page 1951)
Defines a pointer to a function that determines a conversion path for a source and destination encoding, then creates a text encoding converter object and returns a pointer to it.

`TECPluginClearContextInfoPtr` (page 1940)
Defines a pointer to a function that resets a converter object to its initial state.

`TECPluginConvertTextEncodingPtr` (page 1941)
Defines a pointer to a function that converts stream of text from a source encoding to a destination encoding, using the conversion path specified by the converter object you supply.

`TECPluginFlushConversionPtr` (page 1943)
Defines a pointer to a function that flushes out any data in a converter object's temporary buffers and resets the converter object.

`TECPluginDisposeEncodingConverterPtr` (page 1941)
Defines a pointer to a function that disposes of a converter object.

`TECPluginNewEncodingSnifferPtr` (page 1952)
Defines a pointer to a function that creates a sniffer object and returns a reference to it.

`TECPluginClearSnifferContextInfoPtr` (page 1940)
Defines a pointer to a function that resets a sniffer object to its initial settings.

`TECPluginSniffTextEncodingPtr` (page 1952)
Defines a pointer to a function that analyzes a text stream and returns the probable encodings in a ranked list, based on an array of possible encodings you supply; it also returns the number of errors and features for each encoding.

`TECPluginDisposeEncodingSnifferPtr` (page 1942)
Defines a pointer to a function that disposes of a sniffer object.

`TECPluginGetCountAvailableTextEncodingsPtr` (page 1945)
Defines a pointer to a function that obtains the available text encodings.

`TECPluginGetCountAvailableTextEncodingPairsPtr` (page 1944)
Defines a pointer to a function that obtains the available text encoding pairs.

`TECPluginGetCountDestinationTextEncodingsPtr` (page 1946)
Defines a pointer to a function that counts and returns the number of destination encodings to which a specified source encoding can be converted in one step.

`TECPluginGetCountSubTextEncodingsPtr` (page 1947)
Defines a pointer to a function that obtains the text encoding specifications for the subencodings the encoding scheme supports.

`TECPluginGetCountAvailableSniffersPtr` (page 1943)
Defines a pointer to a function that counts and returns the number of sniffers available in all installed plug-ins.

`TECPluginGetCountWebEncodingsPtr` (page 1948)
Defines a pointer to a function that obtains the available web text encodings.

`TECPluginGetCountMailEncodingsPtr` (page 1947)
Defines a pointer to a function that obtains the text encodings available for email.

`TECPluginGetTextEncodingInternetNamePtr` (page 1950)
Defines a pointer to a function that obtains the Internet text encoding name for a text encoding specification.

`TECPluginGetTextEncodingFromInternetNamePtr` (page 1949)
Defines a pointer to a function that obtains the text encoding for an Internet text encoding name.

Callbacks

TECPluginClearContextInfoPtr

Defines a pointer to a function that resets a converter object to its initial state.

```
typedef OSStatus (*TECPluginClearContextInfoPtr)
(
    TECObjectRef encodingConverter,
    TECConverterContextRec * plugContext
);
```

If you name your function `MyTECPluginClearContextInfo`, you would declare it like this:

```
OSStatus MyTECPluginClearContextInfoPtr
(
    TECObjectRef encodingConverter,
    TECConverterContextRec * plugContext
);
```

Parameters

encodingConverter
A reference to the text encoding converter object that needs to be reset.

plugContext
A pointer to a TEC converter context record.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginClearSnifferContextInfoPtr

Defines a pointer to a function that resets a sniffer object to its initial settings.

```
typedef OSStatus (*TECPluginClearSnifferContextInfoPtr)
(
    TECSnifferObjectRef encodingSniffer,
    TECSnifferContextRec * snifContext
);
```

If you name your function `MyTECPluginClearSnifferContextInfo`, you would declare it like this:

```
OSStatus MyTECPluginClearSnifferContextInfoPtr
(
    TECSnifferObjectRef encodingSniffer,
    TECSnifferContextRec * snifContext
);
```

Parameters

encodingSniffer

A reference to the sniffer object that needs to be reset.

snifContext

A pointer to a TEC sniffer context record.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginConvertTextEncodingPtr

Defines a pointer to a function that converts stream of text from a source encoding to a destination encoding, using the conversion path specified by the converter object you supply.

```
typedef OSStatus (*TECPluginConvertTextEncodingPtr)
(
    TECObjectRef encodingConverter,
    TECConverterContextRec * plugContext
);
```

If you name your function `MyTECPluginConvertTextEncoding`, you would declare it like this:

```
OSStatus MyTECPluginConvertTextEncodingPtr
(
    TECObjectRef encodingConverter,
    TECConverterContextRec * plugContext
);
```

Parameters

encodingConverter

A reference to the text encoding converter object to use for the conversion.

plugContext

A pointer to a TEC converter context record that contains the text and other information needed for the conversion.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginDisposeEncodingConverterPtr

Defines a pointer to a function that disposes of a converter object.

```
typedef OSStatus (*TECPluginDisposeEncodingConverterPtr)
(
    TECObjectRef newEncodingConverter,
    TECConverterContextRec * plugContext
);
```

If you name your function `MyTECPluginDisposeEncodingConverter`, you would declare it like this:

```
OSStatus MyTECPluginDisposeEncodingConverterPtr
(
    TECObjectRef newEncodingConverter,
    TECConverterContextRec * plugContext
);
```

Parameters

newEncodingConverter

A reference to the converter object to dispose of.

plugContext

A pointer to a TEC converter context record.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginDisposeEncodingSnifferPtr

Defines a pointer to a function that disposes of a sniffer object.

```
typedef OSStatus (*TECPluginDisposeEncodingSnifferPtr)
(
    TECSnifferObjectRef encodingSniffer,
    TECSnifferContextRec * snifContext
);
```

If you name your function `MyTECPluginDisposeEncodingSniffer`, you would declare it like this:

```
OSStatus MyTECPluginDisposeEncodingSnifferPtr
(
    TECSnifferObjectRef encodingSniffer,
    TECSnifferContextRec * snifContext
);
```

Parameters

encodingSniffer

A reference to the sniffer object you want to dispose.

snifContext

A pointer to a TEC sniffer context record.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

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

Declared In
`TextEncodingPlugin.h`

TECPluginFlushConversionPtr

Defines a pointer to a function that flushes out any data in a converter object's temporary buffers and resets the converter object.

```
typedef OSStatus (*TECPluginFlushConversionPtr)
(
    TECObjectRef encodingConverter,
    TECConverterContextRec * plugContext
);
```

If you name your function `MyTECPluginFlushConversion`, you would declare it like this:

```
OSStatus MyTECPluginFlushConversionPtr
(
    TECObjectRef encodingConverter,
    TECConverterContextRec * plugContext
);
```

Parameters

`encodingConverter`
A reference to the text converter object whose contents are to be flushed.

`plugContext`
A pointer to a TEC converter context record.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

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

Declared In
`TextEncodingPlugin.h`

TECPluginGetCountAvailableSniffersPtr

Defines a pointer to a function that counts and returns the number of sniffers available in all installed plug-ins.

```
typedef OSStatus (*TECPluginGetCountAvailableSniffersPtr)
(
    TextEncoding * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

If you name your function `MyTECPluginGetCountAvailableSniffers`, you would declare it like this:

```
OSStatus MyTECPluginGetCountAvailableSniffersPtr
(
    TextEncoding * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

Parameters

availableEncodings

On return, a pointer to the currently available sniffer text encoding specifications.

maxAvailableEncodings

The number of text encoding specifications the `availableEncodings` array can contain.

actualAvailableEncodings

On the return, the number of text encoding specifications the `availableEncodings` array actually contains.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginGetCountAvailableTextEncodingPairsPtr

Defines a pointer to a function that obtains the available text encoding pairs.

```
typedef OSStatus (*TECPluginGetCountAvailableTextEncodingPairsPtr)
(
    TECConversionInfo * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

If you name your function `MyTECPluginGetCountAvailableTextEncodingPairs`, you would declare it like this:

```
OSStatus MyTECPluginGetCountAvailableTextEncodingPairsPtr
(
    TECConversionInfo * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
```

```
);
```

Parameters

availableEncodings

On return, an array of text encoding conversion information structures, each of which specifies a set of source and destination encodings for a type of conversion.

maxAvailableEncodings

The number of text encoding information structures the `availableEncodings` array can contain.

actualAvailableEncodings

On the return, the number of text encoding information structures the `availableEncodings` array actually contains.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

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

Declared In
`TextEncodingPlugin.h`

TECPluginGetCountAvailableTextEncodingsPtr

Defines a pointer to a function that obtains the available text encodings.

```
typedef OSStatus (*TECPluginGetCountAvailableTextEncodingsPtr)
(
    TextEncoding * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

If you name your function `MyTECPluginGetCountAvailableTextEncodings`, you would declare it like this:

```
OSStatus MyTECPluginGetCountAvailableTextEncodingsPtr
(
    TextEncoding * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

Parameters

availableEncodings

On return, a pointer to the currently available text encoding specifications.

maxAvailableEncodings

The number of text encoding specifications the `availableEncodings` array can contain.

actualAvailableEncodings

On the return, the number of text encoding specifications the `availableEncodings` array actually contains.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

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

Declared In
`TextEncodingPlugin.h`

TECPluginGetCountDestinationTextEncodingsPtr

Defines a pointer to a function that counts and returns the number of destination encodings to which a specified source encoding can be converted in one step.

```
typedef OSStatus (*TECPluginGetCountDestinationTextEncodingsPtr)
(
    TextEncoding inputEncoding,
    TextEncoding * destinationEncodings,
    ItemCount maxDestinationEncodings,
    ItemCount * actualDestinationEncodings
);
```

If you name your function `MyTECPluginGetCountDestinationTextEncodings`, you would declare it like this:

```
OSStatus MyTECPluginGetCountDestinationTextEncodingsPtr
(
    TextEncoding inputEncoding,
    TextEncoding * destinationEncodings,
    ItemCount maxDestinationEncodings,
    ItemCount * actualDestinationEncodings
);
```

Parameters

inputEncoding
The text encoding specification describing the source text.

destinationEncodings
On return, a pointer to text encodings to which the source encoding can be converted in one step.

maxDestinationEncodings
The maximum number of text encodings that can be specified by the `destinationEncodings` parameter.

actualDestinationEncodings
On return, the actual number of text encodings specified by the `destinationEncodings` parameter.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

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

Declared In
`TextEncodingPlugin.h`

TECPluginGetCountMailEncodingsPtr

Defines a pointer to a function that obtains the text encodings available for email.

```
typedef OSStatus (*TECPluginGetCountMailEncodingsPtr)
(
    TextEncoding * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

If you name your function `MyTECPluginGetCountMailEncodings`, you would declare it like this:

```
OSStatus MyTECPluginGetCountMailEncodingsPtr
(
    TextEncoding * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

Parameters

availableEncodings
: On return, a pointer to the text encodings available for email.

maxAvailableEncodings
: The maximum number of text encodings that can be specified by the `availableEncodings` parameter.

actualAvailableEncodings
: On return, the number of text encoding specifications `availableEncodings` actually contains.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginGetCountSubTextEncodingsPtr

Defines a pointer to a function that obtains the text encoding specifications for the subencodings the encoding scheme supports.

```
typedef OSStatus (*TECPluginGetCountSubTextEncodingsPtr)
(
    TextEncoding inputEncoding,
    TextEncoding subEncodings[],
    ItemCount maxSubEncodings,
    ItemCount * actualSubEncodings
);
```

If you name your function `MyTECPluginGetCountSubTextEncodings`, you would declare it like this:

```
OSStatus MyTECPluginGetCountSubTextEncodingsPtr
(
```

```
    TextEncoding inputEncoding,
    TextEncoding subEncodings[],
    ItemCount maxSubEncodings,
    ItemCount * actualSubEncodings
);
```

Parameters

inputEncoding

A text encoding specification.

subEncodings

On return, the array contains the specifications for the subencodings of the `inputEncoding` parameter.

maxSubEncodings

The number of text encoding specifications the `subEncodings` array can contain.

actualSubEncodings

On return, a pointer to number of subencodings in the `subEncodings` array.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginGetCountWebEncodingsPtr

Defines a pointer to a function that obtains the available web text encodings.

```
typedef OSStatus (*TECPluginGetCountWebEncodingsPtr)
(
    TextEncoding * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

If you name your function `MyTECPluginGetCountWebEncodings`, you would declare it like this:

```
OSStatus MyTECPluginGetCountWebEncodingsPtr
(
    TextEncoding * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

Parameters

availableEncodings

On return, points to the currently supported text encodings available for the web.

maxAvailableEncodings

The number of text encodings specifications that `availableEncodings` can specify.

actualAvailableEncodings
On return, the number of text encodings specifications `availableEncodings` actually contains.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

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

Declared In
`TextEncodingPlugin.h`

TECPluginGetPluginDispatchTablePtr

Defines a pointer to a function that returns a pointer to a plug-in dispatch table.

```
typedef TECPluginDispatchTable * (*TECPluginGetPluginDispatchTablePtr)
(
);
```

If you name your function `ConverterPluginGetPluginDispatchTable`, you would declare it like this:

```
TECPluginDispatchTable * ConverterPluginGetPluginDispatchTable();
```

Parameters

Return Value
A pointer to the function dispatch table for the plug-in.

Discussion
You need this callback only for Mac OS X plug-ins. When you create a TEC plug-in in Mac OS X you must export a function named `ConverterPluginGetPluginDispatchTable` with the following prototype:

```
extern TECPluginDispatchTable *ConverterPluginGetPluginDispatchTable (void)
```

This function must return a pointer to the function dispatch table for the plug-in. It is important you name the function `ConverterPluginGetPluginDispatchTable` because `TECPluginGetPluginDispatchTablePtr` is a function pointer to a function of this exact name.

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

Declared In
`TextEncodingPlugin.h`

TECPluginGetTextEncodingFromInternetNamePtr

Defines a pointer to a function that obtains the text encoding for an Internet text encoding name.

```
typedef OSStatus (*TECPluginGetTextEncodingFromInternetNamePtr)
(
    TextEncoding * textEncoding,
    ConstStr255Param encodingName
);
```

If you name your function `MyTECPluginGetTextEncodingFromInternetName`, you would declare it like this:

```
OSStatus MyTECPluginGetTextEncodingFromInternetNamePtr
(
    TextEncoding * textEncoding,
    ConstStr255Param encodingName
);
```

Parameters

textEncoding

On return, a pointer to a structure that contains a text encoding specification for the text encoding name specified by the `encodingName` parameter.

encodingName

An Internet encoding name, in 7-bit US ASCII.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginGetTextEncodingInternetNamePtr

Defines a pointer to a function that obtains the Internet text encoding name for a text encoding specification.

```
typedef OSStatus (*TECPluginGetTextEncodingInternetNamePtr)
(
    TextEncoding textEncoding,
    Str255 encodingName
);
```

If you name your function `MyTECPluginGetTextEncodingInternetName`, you would declare it like this:

```
OSStatus MyTECPluginGetTextEncodingInternetNamePtr
(
    TextEncoding textEncoding,
    Str255 encodingName
);
```

Parameters

textEncoding

A text encoding specification.

encodingName
On return, the Internet encoding name, in 7-bit US ASCII. If there are several Internet encoding names for the same text encoding, the `encodingName` parameter contains the preferred name.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

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

Declared In
`TextEncodingPlugin.h`

TECPluginNewEncodingConverterPtr

Defines a pointer to a function that determines a conversion path for a source and destination encoding, then creates a text encoding converter object and returns a pointer to it.

```
typedef OSStatus (*TECPluginNewEncodingConverterPtr)
(
    TECObjectRef * newEncodingConverter,
    TECConverterContextRec * plugContext,
    TextEncoding inputEncoding,
    TextEncoding outputEncoding
);
```

If you name your function `MyTECPluginNewEncodingConverter`, you would declare it like this:

```
OSStatus MyTECPluginNewEncodingConverterPtr
(
    TECObjectRef * newEncodingConverter,
    TECConverterContextRec * plugContext,
    TextEncoding inputEncoding,
    TextEncoding outputEncoding
);
```

Parameters

newEncodingConverter
A pointer to a converter object. On return, this points to a newly created text converter object.

plugContext
A pointer to a TEC converter context record.

inputEncoding
The text encoding specification for the source text.

outputEncoding
The text encoding specification for the destination text.

Return Value
A result code. See "TEC Manager Result Codes" (page 2026).

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

Declared In
`TextEncodingPlugin.h`

TECPluginNewEncodingSnifferPtr

Defines a pointer to a function that creates a sniffer object and returns a reference to it.

```
typedef OSStatus (*TECPluginNewEncodingSnifferPtr)
(
    TECSnifferObjectRef * encodingSniffer,
    TECSnifferContextRec * snifContext,
    TextEncoding inputEncoding
);
```

If you name your function `MyTECPluginNewEncodingSniffer`, you would declare it like this:

```
OSStatus MyTECPluginNewEncodingSnifferPtr
(
    TECSnifferObjectRef * encodingSniffer,
    TECSnifferContextRec * snifContext,
    TextEncoding inputEncoding
);
```

Parameters

encodingSniffer

A pointer to a sniffer object reference, which is of type `TECSnifferObjectRef` (page 1964). On return, the reference pertains to the newly created sniffer object.

snifContext

A pointer to a TEC sniffer context record.

inputEncoding

The text encoding specification to be detected by the sniffer object.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginSniffTextEncodingPtr

Defines a pointer to a function that analyzes a text stream and returns the probable encodings in a ranked list, based on an array of possible encodings you supply; it also returns the number of errors and features for each encoding.

```
typedef OSStatus (*TECPluginSniffTextEncodingPtr)
(
    TECSnifferObjectRef encodingSniffer,
    TECSnifferContextRec * snifContext
);
```

If you name your function `MyTECPluginSniffTextEncoding`, you would declare it like this:

```
OSStatus MyTECPluginSniffTextEncodingPtr
(
    TECSnifferObjectRef encodingSniffer,
```

```
    TECSnifferContextRec * snifContext
);
```

Parameters

encodingSniffer

A reference to a sniffer object.

snifContext

A pointer to a TEC sniffer context record.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

UnicodeToTextFallbackProcPtr

Defines a pointer to a function that converts a Unicode text element for which there is no destination encoding equivalent in the appropriate mapping table to the fallback character sequence defined by your fallback handler, and returns the converted character sequence to the Unicode Converter.

```
typedef OSStatus (*UnicodeToTextFallbackProcPtr)
(
    UniChar * iSrcUniStr,
    ByteCount iSrcUniStrLen,
    ByteCount * oSrcConvLen,
    TextPtr oDestStr,
    ByteCount iDestStrLen,
    ByteCount * oDestConvLen,
    LogicalAddress iInfoPtr,
    ConstUnicodeMappingPtr iUnicodeMappingPtr
);
```

If you name your function `MyUnicodeToTextFallbackProc`, you would declare it like this:

```
OSStatus MyUnicodeToTextFallbackProcPtr
(
    UniChar * iSrcUniStr,
    ByteCount iSrcUniStrLen,
    ByteCount * oSrcConvLen,
    TextPtr oDestStr,
    ByteCount iDestStrLen,
    ByteCount * oDestConvLen,
    LogicalAddress iInfoPtr,
    ConstUnicodeMappingPtr iUnicodeMappingPtr
);
```

Parameters

iSrcUniStr

A pointer to a single UTF-16 character to be mapped by the fallback handler.

iSrcUniStrLen

The length in bytes of the UTF-16 character indicated by the `iSrcUniStr` parameter. Usually this is 2 bytes, but it could be 4 bytes for a non-BMP character.

oSrcConvLen

On return, a pointer to the length in bytes of the portion of the Unicode character that was actually processed by your fallback handler. Your fallback handler returns this value. It should set this to 0 if none of the text was handled, or 2 or 4 if the Unicode character was handled. This value is initialized to 0 before the fallback handler is called.

oDestStr

A pointer to the output buffer where your handler should place any converted text.

iDestStrLen

The maximum size in bytes of the buffer provided by the `oDestStr` parameter.

oDestConvLen

On return, a pointer to the length in bytes of the fallback character sequence generated by your fallback handler. Your handler should return this length. It is initialized to 0 (zero) before the fallback handler is called.

iInfoPtr

A pointer to a block of memory allocated by your application, which can be used by your fallback handler in any way that you like. This is the same pointer passed as the last parameter of `SetFallbackUnicodeToText` or `SetFallbackUnicodeToTextRun`. How you use the data passed to you in this memory block is particular to your handler. This is similar in use to a reference constant (refcon).

iUnicodeMappingPtr

A constant pointer to a structure of type `UnicodeMapping` (page 1967). This structure identifies a Unicode encoding specification and a particular base encoding specification.

Return Value

A result code. See "TEC Manager Result Codes" (page 2026). Your handler should return `noErr` if it can handle the fallback, or `kTECUnmappableElementErr` if it cannot. It can return other errors for exceptional conditions, such as when the output buffer is too small. If your handler returns `kTECUnmappableElementErr`, then `oSrcConvLen` and `oDestConvLen` are ignored because either the default handler will be called or the default fallback sequence will be used.

Discussion

The Unicode Converter calls your fallback handler when it cannot convert a text string using the mapping table specified by the Unicode converter object passed to either `ConvertFromUnicodeToText` or `ConvertFromUnicodeToPString`. The control flags you set for the `controlFlags` parameter of the function `SetFallbackUnicodeToText` (page 1908) or the `SetFallbackUnicodeToTextRun` (page 1909) stipulate which fallback handler the Unicode Converter should call and which one to try first if both can be used.

When the Unicode Converter calls your handler, it passes to it the Unicode character to be converted and its length, a buffer for the converted string you return and the buffer length, and a pointer to a block of memory containing the data your application supplied to be passed on to your fallback handler.

After you convert the Unicode text segment to fallback characters, you return the fallback character sequence of the converted text in the buffer provided to you and the length in bytes of this fallback character sequence. You also return the length in bytes of the portion of the source Unicode text element that your handler actually processed.

You provide a fallback-handler function for use with the function `CreateUnicodeToTextInfoByEncoding` (page 1893), `ConvertFromUnicodeToPString` (page 1879), `ConvertFromUnicodeToTextRun` (page 1885), or `ConvertFromUnicodeToScriptCodeRun` (page 1880). You associate an application-defined fallback handler with a particular Unicode converter object you intend to pass to the conversion function when you call it.

Text converted from UTF-8 will already have been converted to UTF-16 before the fallback handler is called to process it. Your fallback handler should do all of its processing on text encoded in UTF-16.

Your application-defined fallback handler should not move memory or call any toolbox function that would move memory. If it needs memory, the memory should be allocated before the call to `SetFallbackUnicodeToText` or `SetFallbackUnicodeToTextRun`, and a memory reference should be passed either directly as `iInfoPtr` or in the data referenced by `iInfoPtr`.

To associate a fallback-handler function with a Unicode converter object you use the `SetFallbackUnicodeToText` (page 1908) and `SetFallbackUnicodeToTextRun` (page 1909) functions. For these functions, you must pass a universal procedure pointer (`UniversalProcPtr`). This is derived from a pointer to your function by using the predefined macro `NewUnicodeToTextFallbackProc`.

For versions of the Unicode Converter prior to 1.2, the fallback handler may receive a multiple character text element, so the source string length value could be greater than 2 and the fallback handler may set `srcConvLen` to a value greater than 2. In versions earlier than 1.2.1, the `srcConvLen` and `destConvLen` variables are not initialized to 0; both values are ignored unless the fallback handler returns `noErr`.

The following example shows how to install an application-defined fallback handler. You can name your application-defined fallback handler anything you choose. The name, `MyUnicodeToTextFallbackProc`, used in this example is not significant. However, you must adhere to the parameters, the return type, and the calling convention as expressed in this example, which follows the prototype, because a pointer to this function must be of type `UnicodeToTextFallbackProcPtr` as defined in the `UnicodeConverter.h` header file.

The `UnicodeConverter.h` header file also defines the `UnicodeToTextFallbackUPP` type and the `NewUnicodeToTextFallbackProc` macro.

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

Declared In
`UnicodeConverter.h`

Data Types

ConstScriptCodeRunPtr

Defines a constant script code run pointer.

```
typedef const ScriptCodeRun * ConstScriptCodeRunPtr;
```

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

Declared In
TextCommon.h

ConstTextEncodingRunPtr

Defines a constant text encoding run pointer.

```
typedef const TextEncodingRun * ConstTextEncodingRunPtr;
```

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

Declared In
TextCommon.h

ConstTextPtr

Defines a constant text pointer.

```
typedef const UInt8 * ConstTextPtr;
```

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

Declared In
TextCommon.h

ConstTextToUnicodeInfo

Defines a constant text to Unicode converter object.

```
typedef TextToUnicodeInfo ConstTextToUnicodeInfo;
```

Discussion
The `TruncateForTextToUnicode` (page 1934) function requires a Unicode converter object as a parameter. This function does not modify the contents of the private structure to which the Unicode converter object refers, so it uses the constant Unicode converter object defined by the `ConstTextToUnicodeInfo` data type.

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

Declared In
UnicodeConverter.h

ConstUniCharArrayPtr

Defines a constant Unicode character array pointer.

```
typedef const UniChar * ConstUniCharArrayPtr;
```

Discussion
You specify a constant Unicode character array pointer for Unicode strings used within the scope of a function whose contents are not modified by that function.

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

Declared In
`TextCommon.h`

ConstUnicodeMappingPtr

Defines a constant Unicode mapping pointer.

```
typedef const UnicodeMapping *     ConstUnicodeMappingPtr;
```

Discussion
Many Unicode Converter functions take a pointer to a Unicode mapping structure as a parameter. For functions that do not modify the Unicode mapping contents, the Unicode Converter provides a constant pointer to a Unicode mapping structure defined by the `ConstUnicodeMappingPtr` data type.

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

Declared In
`UnicodeConverter.h`

ConstUnicodeToTextInfo

Defines a constant Unicode to text converter object.

```
typedef UnicodeToTextInfo ConstUnicodeToTextInfo;
```

Discussion
The `TruncateForUnicodeToText` (page 1935) function requires a Unicode converter object as a parameter. This function does not modify the contents of the private structure to which the Unicode converter object refers, so it uses the constant Unicode converter object defined by the `ConstUnicodeToTextInfo` data type.

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

Declared In
`UnicodeConverter.h`

ScriptCodeRun

Contains script code information for a text run.

```
struct ScriptCodeRun {
    ByteOffset offset;
    ScriptCode script;
};
typedef struct ScriptCodeRun ScriptCodeRun;
typedef ScriptCodeRun * ScriptCodeRunPtr;
```

Fields

`offset`

The beginning character position of a text run and its script code in the converted text.

`script`

The script code for the text that begins at the position specified.

Discussion

To return the result of a multiple encoding conversion, the `ConvertFromUnicodeToScriptCodeRun` (page 1880) function uses a script code run structure.

The script code run structure uses an extended script code with values in the range 0–254, which are the text encoding base equivalents to Mac OS encodings. Values 0–32 correspond directly to traditional script codes. This allows a script code run to distinguish Icelandic, Turkish, Symbol, Zapf Dingbats, and so on.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextCommon.h`

TECBufferContextRec

Contains buffers for text and text encoding runs.

```
struct TECBufferContextRec {
    TextPtr textInputBuffer;
    TextPtr textInputBufferEnd;
    TextPtr textOutputBuffer;
    TextPtr textOutputBufferEnd;
    TextEncodingRunPtr encodingInputBuffer;
    TextEncodingRunPtr encodingInputBufferEnd;
    TextEncodingRunPtr encodingOutputBuffer;
    TextEncodingRunPtr encodingOutputBufferEnd;
};
typedef struct TECBufferContextRec TECBufferContextRec;
```

Discussion

This structure is used in the `TECConverterContextRec` (page 1959) data structure that is used for a TEC plug-in.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECConversionInfo

Contains text encoding conversion information.

```
struct TECConversionInfo {
    TextEncoding sourceEncoding;
    TextEncoding destinationEncoding;
    UInt16 reserved1;
    UInt16 reserved2;
};
typedef struct TECConversionInfo TECConversionInfo;
```

Fields

`sourceEncoding`
The text encoding specification for the source text.

`destinationEncoding`
The text encoding specification for the destination text.

`reserved1`
Reserved.

`reserved2`
Reserved.

Discussion

When you call the function `TECGetDirectTextEncodingConversions` (page 1927), you pass an array of text encoding conversion information structures. The function fills these structures with information about each type of supported conversion.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingConverter.h`

TECConverterContextRec

Contains converter information used by a Text Encoding Converter plug-in.

```
struct TECConverterContextRec {
    Ptr pluginRec;
    TextEncoding sourceEncoding;
    TextEncoding destEncoding;
    UInt32 reserved1;
    UInt32 reserved2;
    TECBufferContextRec bufferContext;
    UInt32 contextRefCon;
    ProcPtr conversionProc;
    ProcPtr flushProc;
    ProcPtr clearContextInfoProc;
    UInt32 options1;
    UInt32 options2;
    TECPluginStateRec pluginState;
};
typedef struct TECConverterContextRec TECConverterContextRec;
```

Fields

`pluginRec`

`sourceEncoding`

The text encoding specification for the source text.

`destEncoding`

The text encoding specification for the destination text.

`reserved1`

Reserved.

`reserved2`

Reserved.

`bufferContext`

`contextRefCon`

A 32-bit value containing or referring to plug-in-specific data.

`conversionProc`

A pointer to a callback for your conversion procedure.

`flushProc`

A pointer to a callback for your reset procedure.

`clearContextInfoProc`

A pointer to a callback for our clear procedure.

`options1`

A 32-bit value that specfies options needed by your plug-in.

`options2`

A 32-bit value that specfies options needed by your plug-in.

`pluginState`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECInfo

Contains information about the Unicode Converter, the Text Encoding Converter, and Basic Text Types.

```
struct TECInfo {
    UInt16 format;
    UInt16 tecVersion;
    UInt32 tecTextConverterFeatures;
    UInt32 tecUnicodeConverterFeatures;
    UInt32 tecTextCommonFeatures;
    Str31 tecTextEncodingsFolderName;
    Str31 tecExtensionFileName;
    UInt16 tecLowestTEFileVersion;
    UInt16 tecHighestTEFileVersion;
};
typedef struct TECInfo TECInfo;
typedef TECInfo * TECInfoPtr;
```

Fields

`format`

The current format of the returned structure. The format of the structure is indicated by the `kTECInfoCurrentFormat` constant. Any future changes to the format will always be backwardly compatible; any new fields will be added to the end of the structure.

`tecVersion`

The current version of the Text Encoding Conversion Manager extension in BCD (binary coded decimal), with the first byte indicating the major version; for example, 0x0121 for 1.2.1.

`tecTextConverterFeatures`

New features or bug fixes in the Text Encoding Converter. No bits are currently defined.

`tecUnicodeConverterFeatures`

Bit flags indicating new features or bug fixes in the Unicode Converter. See "Unicode Converter Flags" (page 1977) for the currently defined bit flags.

`tecTextCommonFeatures`

Bit flags indicating new features or bug fixes in Basic Text Types (the Text Common static library). No bits are currently defined.

`tecTextEncodingsFolderName`

A Pascal string with the (possibly localized) name of the Text Encodings folder.

`tecExtensionFileName`

A Pascal string with the (possibly localized) name of the Text Encoding Conversion Manager extension file.

`tecLowestTEFileVersion`

`tecHighestTEFileVersion`

Discussion

The converter information structure is used by the function `TECGetInfo` (page 1929) to hold returned information about the Unicode Converter, the Text Encoding Converter, and Basic Text Types.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextCommon.h`

TECObjectRef

Defines an opaque reference to a converter object.

```
typedef struct OpaqueTECObjectRef * TECObjectRef;
```

Discussion
When making a text conversion, the Text Encoding Converter requires a reference to a converter object that indicates how to accomplish the conversion. Functions, such as `TECCreateConverter` (page 1918), that create a converter object return this reference, which you can then pass to other functions when converting text. A converter object reference is defined by the `TECObjectRef` data type.

The structure of the `OpaqueTECObjectRef` data type is private, and a converter object is not accessible directly.

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

Declared In
`TextEncodingConverter.h`

TECPluginDispatchTable

Contains version and signature information and pointers to the callback functions used by a text encoding converter plug-in.

```
struct TECPluginDispatchTable {
    TECPluginVersion version;
    TECPluginVersion compatibleVersion;
    TECPluginSignature PluginID;
    TECPluginNewEncodingConverterPtr PluginNewEncodingConverter;
    TECPluginClearContextInfoPtr PluginClearContextInfo;
    TECPluginConvertTextEncodingPtr PluginConvertTextEncoding;
    TECPluginFlushConversionPtr PluginFlushConversion;
    TECPluginDisposeEncodingConverterPtr PluginDisposeEncodingConverter;
    TECPluginNewEncodingSnifferPtr PluginNewEncodingSniffer;
    TECPluginClearSnifferContextInfoPtr PluginClearSnifferContextInfo;
    TECPluginSniffTextEncodingPtr PluginSniffTextEncoding;
    TECPluginDisposeEncodingSnifferPtr PluginDisposeEncodingSniffer;
   TECPluginGetCountAvailableTextEncodingsPtr PluginGetCountAvailableTextEncodings;
    TECPluginGetCountAvailableTextEncodingPairsPtr
PluginGetCountAvailableTextEncodingPairs;
    TECPluginGetCountDestinationTextEncodingsPtr
PluginGetCountDestinationTextEncodings;
    TECPluginGetCountSubTextEncodingsPtr PluginGetCountSubTextEncodings;
    TECPluginGetCountAvailableSniffersPtr PluginGetCountAvailableSniffers;
    TECPluginGetCountWebEncodingsPtr PluginGetCountWebTextEncodings;
    TECPluginGetCountMailEncodingsPtr PluginGetCountMailTextEncodings;
    TECPluginGetTextEncodingInternetNamePtr PluginGetTextEncodingInternetName;
    TECPluginGetTextEncodingFromInternetNamePtr
PluginGetTextEncodingFromInternetName;
};
typedef struct TECPluginDispatchTable TECPluginDispatchTable;
```

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

Declared In
TextEncodingPlugin.h

TECPluginSig

Defines a data type for a Text Encoding Converter plug-in signature.

```
typedef OSType TECPluginSig;
```

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

Declared In
TextEncodingConverter.h

TECPluginSignature

Defines a data type for a Text Encoding Converter plug-in signature.

```
typedef OSType TECPluginSignature;
```

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

Declared In
TextEncodingConverter.h

TECPluginStateRec

Contains state information for a Text Encoding Converter plug-in.

```
struct TECPluginStateRec {
    UInt8 state1;
    UInt8 state2;
    UInt8 state3;
    UInt8 state4;
    UInt32 longState1;
    UInt32 longState2;
    UInt32 longState3;
    UInt32 longState4;
};
typedef struct TECPluginStateRec TECPluginStateRec;
```

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

Declared In
TextEncodingPlugin.h

TECPluginVersion

Defines a data type for Text Encoding Converter plug-in version.

```
typedef UInt32 TECPluginVersion;
```

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

Declared In
`TextEncodingConverter.h`

TECSnifferContextRec

Contains infomation used by a sniffer object.

```
struct TECSnifferContextRec {
    Ptr pluginRec;
    TextEncoding encoding;
    ItemCount maxErrors;
    ItemCount maxFeatures;
    TextPtr textInputBuffer;
    TextPtr textInputBufferEnd;
    ItemCount numFeatures;
    ItemCount numErrors;
    UInt32 contextRefCon;
    ProcPtr sniffProc;
    ProcPtr clearContextInfoProc;
    TECPluginStateRec pluginState;
};
typedef struct TECSnifferContextRec TECSnifferContextRec;
```

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

Declared In
`TextEncodingPlugin.h`

TECSnifferObjectRef

Defines a reference to an opaque sniffer object.

```
typedef struct OpaqueTECSnifferObjectRef * TECSnifferObjectRef;
```

Discussion
When analyzing text for possible encodings, the Text Encoding Converter requires a reference to a sniffer object that specifies what types of encodings can be detected. You receive this reference when calling the function `TECCreateSniffer` (page 1920). A sniffer object reference is defined by the `TECSnifferObjectRef` data type. The structure of the `OpaqueTECObjectRef` data type is private, and a sniffer object is not accessible directly.

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

Declared In
`TextEncodingConverter.h`

TextEncoding

Defines a data type for a text encoding value.

```
typedef UInt32 TextEncoding;
```

Discussion
A `TextEncoding` value is specified by a text encoding base, a text encoding variant, and a text encoding format. You can obtain a `TextEncoding` value by calling the function `CreateTextEncoding` (page 1890). When you call this function, you can provide the `TextEncodingBase`, `TextEncodingVariant`, and `TextEncodingFormat` data types.

A `TextEncoding` value is used, for example, to identify the encoding of text passed to a text converter. Two `TextEncoding` values are needed—for source and destination encoding—when calling the Text Encoding Converter or the Unicode Converter to convert text.

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

Declared In
`TextCommon.h`

TextEncodingRun

Contains text encoding information for a text run.

```
struct TextEncodingRun {
    ByteOffset offset;
    TextEncoding textEncoding;
};
typedef struct TextEncodingRun TextEncodingRun;
typedef TextEncodingRun * TextEncodingRunPtr;
```

Fields
`offset`
The beginning character position of a run of text in the converted text string.
`textEncoding`
The encoding of the text run that begins at the position specified.

Discussion
It is not always possible to convert text expressed in Unicode to another single encoding because no other single encoding encompasses the Unicode character encoding range. To adjust for this, you can create a Unicode mapping structure array that specifies the target encodings the Unicode text should be converted to when multiple encodings must be used.

If the `kUnicodeTextRunMask` flag is set, `ConvertFromUnicodeToTextRun` (page 1885) and `ConvertFromUnicodeToScriptCodeRun` (page 1880) may convert Unicode text to a string of text containing multiple text encoding runs. Each run contains text expressed in a different encoding from that of the preceding or following text segment. For each text encoding run in the string, a `TextEncodingRun` structure indicates the beginning offset and the text encoding for that run.

Functions that convert text from Unicode to a text run return the converted text in an array of text encoding run structures. A text encoding run structure is defined by the `TextEncodingRun` data type.

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

Declared In
`TextCommon.h`

TextEncodingVariant

Defines a data type for a text encoding variant.

```
typedef UInt32 TextEncodingVariant;
```

Discussion
The following enumerations define text encoding variant constants: "Encoding Variants for Big-5" (page 1988), "Encoding Variants for MacArabic" (page 1989), "Encoding Variants for MacCroatian" (page 1990), "Encoding Variants for MacCyrillic" (page 1991), "Encoding Variants for MacFarsi" (page 1991), "Encoding Variants for MacHebrew" (page 1992), "Encoding Variants for MacIcelandic" (page 1992), "Encoding Variants for MacJapanese" (page 1993), "Encoding Variants for MacRoman Related to Currency" (page 1996), "Encoding Variants for MacRomanian" (page 1997), "Encoding Variants for MacRomanLatin1" (page 1997), "Encoding Variants for MacRoman" (page 1994), and "Encoding Variants for MacVT100" (page 1998).

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

Declared In
`TextCommon.h`

TextToUnicodeInfo

Defines reference to an opaque Unicode converter object.

```
typedef struct OpaqueTextToUnicodeInfo * TextToUnicodeInfo;
```

Discussion
A Unicode converter object is a private object containing mapping and state information. Many of the Unicode Converter functions that perform conversions require a Unicode converter object containing information used for the conversion process. There are three types of Unicode converter objects, all serving the same purpose but used for different types of conversions. You use the `TextToUnicodeInfo` type, described here, for converting from non-Unicode text to Unicode text.

Because your application cannot directly create or modify the contents of the private Unicode converter object, the Unicode Converter provides functions to create and dispose of it. To create a Unicode converter object for converting from non-Unicode text to Unicode text, your application must first call either the function `CreateTextToUnicodeInfo` (page 1890) or the function `CreateTextToUnicodeInfoByEncoding` (page 1891) to provide the mapping information required for the conversion. You can then pass this object to the function `ConvertFromTextToUnicode` (page 1877) or `ConvertFromPStringToUnicode` (page 1876) to identify the information to be used in performing the actual conversion. After you have finished using the object, you should release the memory allocated for it by calling the function `DisposeTextToUnicodeInfo` (page 1897). The `TextToUnicodeInfo` data type defines the Unicode converter object.

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

Declared In
UnicodeConverter.h

UniCharArrayOffset

Represents the boundary between two characters.

```
typedef UInt32 UniCharArrayOffset;
```

Discussion
A UniCharArrayOffset represents the boundary between two characters. For example, the first character in a buffer lies between offsets 0 and 1. So the first character in the buffer can be referred to as either "offset 0, leading" or "offset 1, trailing." This distinction is useful when you deal with caret positions.

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

Declared In
TextCommon.h

UnicodeMapping

Contains information for mapping to or from Unicode encoding.

```
struct UnicodeMapping {
    TextEncoding unicodeEncoding;
    TextEncoding otherEncoding;
    UnicodeMapVersion mappingVersion;
};
typedef struct UnicodeMapping UnicodeMapping;
typedef UnicodeMapping * UnicodeMappingPtr;
```

Fields
unicodeEncoding
: A Unicode text encoding specification of type TextEncoding.

otherEncoding
: A text encoding specification for the text to be converted to or from Unicode.

mappingVersion
: The version of the Unicode mapping table to be used.

Discussion
A Unicode mapping structure contains a complete text encoding specification for a Unicode encoding, a complete non-Unicode text encoding specification giving the encoding for the text to be converted to or from Unicode, and the version of the mapping table to be used for conversion. You use a structure of this type to specify the text encodings to and from which the text string is to be converted. A Unicode mapping structure is defined by the UnicodeMapping data type.

You can specify a variety of normalization options by setting up the Unicode mapping structure as described in the following.

To specify normal canonical decomposition according to Unicode 3.2 rules, with no exclusions ("Canonical decomposition 3.2"), set up the UnicodeMapping structure as follows:

```
mapping.unicodeEncoding (in) = Unicode 2.x-3.x, kUnicodeNoSubset,
kUnicode16BitFormat
mapping.otherEncoding (out) = Unicode 2.x-3.x, kUnicodeCanonicalDecompVariant,
  kUnicode16BitFormat
mapping.mappingVersion = kUnicodeUseLatestMapping
```

Examples:

```
u00E0 -> u0061 + u0300
u0061 + u0300 -> u0061 + u0300
u03AC ->  u03B1 + u0301 (3.2 rules)
uF900 -> u8C48
u00E0 + u0323 -> u0061 + u0323 + u0300  (correct)
```

To specify canonical decomposition according to Unicode 3.2 rules, with HFS+ exclusions ("HFS+ decomposition 3.2"), set up the `UnicodeMapping` structure in one of the following ways. The second method is for compatibility with the old method of using `mappingVersion = kUnicodeUseHFSPlusMapping`.

```
// Method 1
mapping.unicodeEncoding (in) = Unicode 2.x-3.x, kUnicodeNoSubset,
kUnicode16BitFormat
mapping.otherEncoding (out) = Unicode 2.x-3.x, kUnicodeHFSPlusDecompVariant,
kUnicode16BitFormat
mapping.mappingVersion = kUnicodeUseLatestMapping
// Method 2
mapping.unicodeEncoding (in) = Unicode 2.x-3.x, kUnicode16BitFormat,
kUnicode16BitFormat
mapping.otherEncoding (out) = Unicode 2.x, kUnicodeCanonicalDecompVariant,
kUnicode16BitFormat
mapping.mappingVersion = kUnicodeUseHFSPlusMapping
```

Examples:

```
u00E0 -> u0061 + u0300
u0061 + u0300 -> u0061 + u0300
u03AC ->  u03B1 + u0301 (3.2 rules)
uF900 -> uF900 (decomposition excluded for HFS+)
u00E0 + u0323 -> u0061 + u0323 + u0300  (correct)
```

To specify normal canonical composition according to Unicode 3.2 rules, set up the `UnicodeMapping` structure as follows:

```
mapping.unicodeEncoding (in) = Unicode 2.x-3.x, kUnicodeNoSubset,
kUnicode16BitFormat
mapping.otherEncoding (out) = Unicode 2.x-3.x, kUnicodeCanonicalCompVariant,
kUnicode16BitFormat
mapping.mappingVersion = kUnicodeUseLatestMapping
```

Examples:

```
u00E0 -> u00E0
u0061 + u0300 -> u00E0
u03AC ->  u03AC
uF900 -> u8C48
u00E0 + u0323 -> u1EA1 u0300 (correct)
```

To specify canonical composition according to Unicode 3.2 rules, but using the HFS+ decomposition exclusions, set up the `UnicodeMapping` structure as follows. This is the form to use if you want to obtain a composed form that dervide from the decomposed form used for HFS+ filenames.

```
mapping.unicodeEncoding (in) = Unicode 2.x-3.x, kUnicodeNoSubset,
kUnicode16BitFormat
mapping.otherEncoding (out) = Unicode 2.x-3.x, kUnicodeHFSPlusCompVariant,
kUnicode16BitFormat
mapping.mappingVersion = kUnicodeUseLatestMapping
```

Examples:

```
u00E0 -> u00E0
u0061 + u0300 -> u00E0
u03AC ->  u03AC
uF900 -> uF900
u00E0 + u0323 -> u1EA1 u0300 (correct)
```

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

Declared In
`UnicodeConverter.h`

UnicodeToTextFallbackUPP

Defines a universal procedure pointer to a Unicode-to-text-fallback callback function.

```
typedef UnicodeToTextFallbackProcPtr UnicodeToTextFallbackUPP;
```

Discussion
For more information, see the description of the `UnicodeToTextFallbackProcPtr` (page 1953) callback function.

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

Declared In
`UnicodeConverter.h`

UnicodeToTextInfo

Defines a reference to an opaque Unicode to text converter object.

```
typedef struct OpaqueUnicodeToTextInfo * UnicodeToTextInfo;
```

Discussion
Many of the Unicode Converter functions that perform conversions require a Unicode converter object containing information used for the conversion process. There are three types of Unicode converter objects used for different types of conversions. You use the `UnicodeToTextInfo` type, described here, for converting from Unicode to text.

Because your application cannot directly create or modify the contents of the private Unicode converter object, the Unicode Converter provides functions to create and dispose of it. To create a Unicode converter object for converting from Unicode to text, your application must first call either the function `CreateUnicodeToTextInfo` (page 1892) or `CreateUnicodeToTextInfoByEncoding` (page 1893).

You can then pass this object to the function `ConvertFromUnicodeToText` (page 1883) or `ConvertFromUnicodeToPString` (page 1879) to identify the information used to perform the actual conversion. After you have finished using the object, you should release the memory allocated for it by calling the function `DisposeUnicodeToTextInfo` (page 1898).

A Unicode converter object for this purpose is defined by the `UnicodeToTextInfo` data type.

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

Declared In
`UnicodeConverter.h`

UnicodeToTextRunInfo

Defines a reference to an opaque Unicode to text run information converter object.

```
typedef struct OpaqueUnicodeToTextRunInfo * UnicodeToTextRunInfo;
```

Discussion
Many of the Unicode Converter functions that perform conversions require a Unicode converter object containing information used for the conversion process. There are three types of Unicode converter objects used for different types of conversions. You use the `UnicodeToTextRunInfo` type, described here, for converting from Unicode to multiple encodings.

Because your application cannot directly create or modify the contents of the private Unicode converter object, the Unicode Converter provides functions to create and dispose of it. You can use any of three functions to create a Unicode converter object for converting from Unicode to multiple encodings. You can use `CreateUnicodeToTextRunInfo` (page 1894), `CreateUnicodeToTextRunInfoByEncoding` (page 1895), or `CreateUnicodeToTextRunInfoByScriptCode` (page 1896).

You can then pass this object to the function `ConvertFromUnicodeToTextRun` (page 1885) or `ConvertFromUnicodeToScriptCodeRun` (page 1880) to identify the information used to perform the actual conversion. After you have finished using the object, you should release the memory allocated for it by calling the function `DisposeUnicodeToTextRunInfo` (page 1898).

A Unicode converter object for this purpose is defined by the `UnicodeToTextRunInfo` data type.

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

Declared In
`UnicodeConverter.h`

Constants

Feature Selectors

Conversion Flags

Specify how to perform conversion of text from one encoding to another.

```
enum {
    kUnicodeUseFallbacksBit = 0,
    kUnicodeKeepInfoBit = 1,
    kUnicodeDirectionalityBits = 2,
    kUnicodeVerticalFormBit = 4,
    kUnicodeLooseMappingsBit = 5,
    kUnicodeStringUnterminatedBit = 6,
    kUnicodeTextRunBit = 7,
    kUnicodeKeepSameEncodingBit = 8,
    kUnicodeForceASCIIRangeBit = 9,
    kUnicodeNoHalfwidthCharsBit = 10,
    kUnicodeTextRunHeuristicsBit = 11,
    kUnicodeMapLineFeedToReturnBit = 12
};
```

Constants

`kUnicodeUseFallbacksBit`
Enables use of fallback mappings.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeKeepInfoBit`
Sets the keep-information control flag.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeDirectionalityBits`
Sets directionality.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeVerticalFormBit`
Sets the vertical form control flag.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeLooseMappingsBit`
Enables use of the loose-mapping portion of a character mapping table.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeStringUnterminatedBit`
Sets the string-unterminated control flag.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeTextRunBit`
Sets the text-run control flag.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeKeepSameEncodingBit`
Sets the keep-same-encoding control flag.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeForceASCIIRangeBit`
Sets the force ASCII range control flag.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeNoHalfwidthCharsBit`
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeTextRunHeuristicsBit`
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeMapLineFeedToReturnBit`
Available in Mac OS X v10.2 and later.
Declared in `UnicodeConverter.h`.

Conversion Masks

Set or text for conversion flags.

```
enum {
    kUnicodeUseFallbacksMask = 1L << kUnicodeUseFallbacksBit,
    kUnicodeKeepInfoMask = 1L << kUnicodeKeepInfoBit,
    kUnicodeDirectionalityMask = 3L << kUnicodeDirectionalityBits,
    kUnicodeVerticalFormMask = 1L << kUnicodeVerticalFormBit,
    kUnicodeLooseMappingsMask = 1L << kUnicodeLooseMappingsBit,
    kUnicodeStringUnterminatedMask = 1L << kUnicodeStringUnterminatedBit,
    kUnicodeTextRunMask = 1L << kUnicodeTextRunBit,
    kUnicodeKeepSameEncodingMask = 1L << kUnicodeKeepSameEncodingBit,
    kUnicodeForceASCIIRangeMask = 1L << kUnicodeForceASCIIRangeBit,
    kUnicodeNoHalfwidthCharsMask = 1L << kUnicodeNoHalfwidthCharsBit,
    kUnicodeTextRunHeuristicsMask = 1L << kUnicodeTextRunHeuristicsBit,
    kUnicodeMapLineFeedToReturnMask = 1L << kUnicodeMapLineFeedToReturnBit
};
```

Constants

`kUnicodeUseFallbacksMask`

A mask for setting the Unicode-use-fallbacks conversion flag. The Unicode Converter uses fallback mappings when it encounters a source text element for which there is no equivalent destination encoding. Fallback mappings are mappings that do not preserve the meaning or identity of the source character but represent a useful approximation of it. See the function `SetFallbackUnicodeToText` (page 1908).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeKeepInfoMask`

A mask for setting the keep-information control flag which governs whether the Unicode Converter keeps the current state stored in the Unicode converter object before converting the text string.

If you clear this flag, the converter will initialize the Unicode converter object before converting the text string and assume that subsequent calls do not need any context, such as direction state for the current call.

If you set the flag, the converter uses the current state. This is useful if your application must convert a stream of text in pieces that are not block delimited. You should set this flag for each call in a series of calls on the same text stream.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeDirectionalityMask`

A mask for setting the directionality control flag

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeVerticalFormMask`

A mask for setting the vertical form control flag. The vertical form control flag tells the Unicode Converter how to map text elements for which there are both abstract and vertical presentation forms in the destination encoding.

If set, the converter maps these text elements to their vertical forms, if they are available.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeLooseMappingsMask`

A mask that determines whether the Unicode Converter should use the loose-mapping portion of a mapping table for character mapping if the strict mapping portion of the table does not include a destination encoding equivalent for the source text element.

If you clear this flag, the converter will use only the strict equivalence portion.

If set this flag and a conversion for the source text element does not exist in the strict equivalence portion of the mapping table, then the converter uses the loose mapping section.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeStringUnterminatedMask`

A mask for setting the string-unterminated control flag. Determines how the Unicode Converter handles text-element boundaries and direction resolution at the end of an input buffer.

If you clear this bit, the converter treats the end of the buffer as the end of text.

If you set this bit, the converter assumes that the next call you make using the current context will supply another buffer of text that should be treated as a continuation of the current text. For example, if the last character in the input buffer is `'A'`, `ConvertFromUnicodeToText` stops conversion at the `'A'` and returns `kTECIncompleteElementErr`, because the next buffer could begin with a combining diacritical mark that should be treated as part of the same text element. If the last character in the input buffer is a control character, `ConvertFromUnicodeToText` does not return `kTECIncompleteElementErr` because a control character could not be part of a multiple character text element.

In attempting to analyze the text direction, when the Unicode Converter reaches the end of the current input buffer and the direction of the current text element is still unresolved, if you clear this flag, the converter treats the end of the buffer as a block separator for direction resolution. If you set this flag, it sets the direction as undetermined

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeTextRunMask`

A mask for setting the text-run control flag which determines how the Unicode Converter converts Unicode text to a non-Unicode encoding when more than one possible destination encoding exists.

If you clear this flag, the function `ConvertFromUnicodeToTextRun` (page 1885) or `ConvertFromUnicodeToScriptCodeRun` (page 1880) attempts to convert the Unicode text to the single encoding from the list of encodings in the Unicode converter object that produces the best result, that is, that provides for the greatest amount of source text conversion.

If you set this flag, `ConvertFromUnicodeToTextRun` or `ConvertFromUnicodeToScriptCodeRun`, which are the only functions to which it applies, may generate a destination string that combines text in any of the encodings specified by the Unicode converter object.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeKeepSameEncodingMask`

A mask for setting the keep-same-encoding control flag. Determines how the Unicode Converter treats the conversion of Unicode text following a text element that could not be converted to the first destination encoding when multiple destination encodings exist. This control flag applies only if the `kUnicodeTextRunMask` control flag is set.

If you set this flag, the function `ConvertFromUnicodeToTextRun` (page 1885) attempts to minimize encoding changes in the conversion of the source text string; that is, once it is forced to make an encoding change, it attempts to use that encoding as the conversion destination for as long as possible.

If you clear this flag, `ConvertFromUnicodeToTextRun` attempts to keep most of the converted string in one encoding, switching to other encodings only when necessary.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeForceASCIIRangeMask`

A mask for setting the force ASCII range control flag. If an encoding normally treats 1-byte code points `0x00–0x7F` as an ISO 646 national variant that is different from ASCII, setting this flag forces `0x00–0x7F` to be treated as ASCII. For example, Japanese encodings such as Shift-JIS generally treat `0x00–0x7F` as JIS Roman, with `0x5C` as YEN SIGN instead of REVERSE SOLIDUS, but when converting a DOS file path you may want to set this flag so that `0x5C` is mapped as REVERSE SOLIDUS.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeNoHalfwidthCharsMask`

Sets the no halfwidth characters control flag.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeTextRunHeuristicsMask`

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMapLineFeedToReturnMask`

Specifies mapping o the LF (LineFeed) character used in Unix to represent new lines to the CR (CarriageReturn) used in Mac encodings. This option has an effect only when used with the constant `kUnicodeLooseMappingsMask`. You can pass both constants as `iControlFlags` parameters to the functions `ConvertFromUnicodeToText`, `ConvertFromUnicodeToTextRun`, and `ConvertFromUnicodeToScriptCodeRun`.

Available in Mac OS X v10.2 and later.

Declared in `UnicodeConverter.h`.

Discussion

You use these constants to specify how the conversion of text from one encoding to another is performed. You use these masks as the `controlFlags` parameter in the `ConvertFromTextToUnicode` (page 1877), `ConvertFromUnicodeToText` (page 1883), `ConvertFromUnicodeToScriptCodeRun` (page 1880), `ConvertFromUnicodeToTextRun` (page 1885), and `TruncateForUnicodeToText` (page 1935) functions. A different subset of control masks applies to each of these functions. Using the bitmask constants, you can perform a bitwise `OR` operation to set the pertinent flags for a particular function's parameters. For example, when you call a function, you might pass the following `controlFlags` parameter setting:

```
controlflags=kUnicodeUseFallbacksMask |kUnicodeLooseMappingsMask;
```

Directionality Flags

Specify a text direction.

```
enum {
    kUnicodeDefaultDirection = 0,
    kUnicodeLeftToRight = 1,
    kUnicodeRightToLeft = 2
};
```

Constants

`kUnicodeDefaultDirection`
Use the default direction.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeLeftToRight`
Indicates left to right direction.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeRightToLeft`
Indicates right to left direction.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

Directionality Masks

Set or text for directionality bits.

```
enum {
    kUnicodeDefaultDirectionMask = kUnicodeDefaultDirection <<
kUnicodeDirectionalityBits,
    kUnicodeLeftToRightMask = kUnicodeLeftToRight << kUnicodeDirectionalityBits,
    kUnicodeRightToLeftMask = kUnicodeRightToLeft << kUnicodeDirectionalityBits
};
```

Constants

`kUnicodeDefaultDirectionMask`
A mask for setting the global, or base, line direction for the text being converted. The value `kUnicodeDefaultDirectionMask` tells the converter to use the value of the first strong direction character in the string. This determines which direction the converter should use for resolution of neutral coded characters, such as spaces that occur between sets of coded characters having different directions—for example, between Latin and Arabic characters—rendering ambiguous the direction of the space character.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeLeftToRightMask`
: A mask for setting the global, or base, line direction for the text being converted. The value `kUnicodeLeftToRightMask` tells the converter that the base paragraph direction is left to right. This determines which direction the converter should use for resolution of neutral coded characters, such as spaces that occur between sets of coded characters having different directions—for example, between Latin and Arabic characters—rendering ambiguous the direction of the space character.

  Available in Mac OS X v10.0 and later.

  Declared in `UnicodeConverter.h`.

`kUnicodeRightToLeftMask`
: The value `kUnicodeRightToLeftMask` tells the converter that the base paragraph direction is right to left. This determines which direction the converter should use for resolution of neutral coded characters, such as spaces that occur between sets of coded characters having different directions—for example, between Latin and Arabic characters—rendering ambiguous the direction of the space character.

  Available in Mac OS X v10.0 and later.

  Declared in `UnicodeConverter.h`.

Unicode Converter Flags

Specify features for bug fixes in the Unicode Converter.

```
enum {
    kTECKeepInfoFixBit = 0,
    kTECFallbackTextLengthFixBit = 1,
    kTECTextRunBitClearFixBit = 2,
    kTECTextToUnicodeScanFixBit = 3,
    kTECAddForceASCIIChangesBit = 4,
    kTECPreferredEncodingFixBit = 5,
    kTECAddTextRunHeuristicsBit = 6,
    kTECAddFallbackInterruptBit = 7
};
```

Constants

`kTECKeepInfoFixBit`
: This is set if the Unicode Converter has a bug fix to stop ignoring certain control flags

  Available in Mac OS X v10.0 and later.

  Declared in `TextCommon.h`.

`kTECFallbackTextLengthFixBit`
: This is set if the Unicode Converter has a bug fix to use the source length (`srcConvLen`) and destination length (`destConvLen`) returned by a caller-supplied fall-back handler for any status it returns except `kTECUnmappableElementErr`. Previously it honored only these values if `noErr` was returned.

  Available in Mac OS X v10.0 and later.

  Declared in `TextCommon.h`.

`kTECTextRunBitClearFixBit`
: This is set if `ConvertFromUnicodeToTextRun` and `ConvertFromUnicodeToScriptCodeRun` function correctly if the `kUnicodeTextRunBit` is clear.

  Available in Mac OS X v10.0 and later.

  Declared in `TextCommon.h`.

`kTECTextToUnicodeScanFixBit`
: This is set if `ConvertFromTextToUnicode` is enhanced so mappings can depend on context and saved state. The consequences of this are (1) malformed input results in `kTextMalformedInputErr`; (2) `ConvertFromTextToUnicode` accepts the control flags `kUnicodeLooseMappingsMask`, `kUnicodeKeepInfoMask`, and `kUnicodeStringUnterminatedMask`; (3) elimination of redundant direction overrides when converting Mac OS Arabic and Hebrew to Unicode; and (4) improved mapping of 0x30-0x39 digits in Mac OS Arabic when loose mappings are used.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTECAddForceASCIIChangesBit`
: This is set if the new control flag bits `kUnicodeForceASCIIRangeBit` and `kUnicodeNoHalfwidthCharsBit` are supported for use with the functions `ConvertFromTextToUnicode`, `ConvertFromUnicodeToText`, and so forth.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTECPreferredEncodingFixBit`
: This is set to indicate that if a preferred encoding is specified for `CreateUnicodeToTextRunInfo` and related functions, they handle it correctly even if it does not match the system script.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTECAddTextRunHeuristicsBit`
: Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTECAddFallbackInterruptBit`
: Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Discussion
These are bit flags to indicate new features for bug fixes in the Unicode Converter. They are used by the `TECInfo` (page 1961) data type.

Unicode Converter Masks

Set or test for Unicode converter flags.

```
enum {
  kTECKeepInfoFixMask           = 1L << kTECKeepInfoFixBit,
  kTECFallbackTextLengthFixMask = 1L << kTECFallbackTextLengthFixBit,
  kTECTextRunBitClearFixMask    = 1L << kTECTextRunBitClearFixBit,
  kTECTextToUnicodeScanFixMask  = 1L << kTECTextToUnicodeScanFixBit,
  kTECAddForceASCIIChangesMask  = 1L << kTECAddForceASCIIChangesBit,
  kTECPreferredEncodingFixMask  = 1L << kTECPreferredEncodingFixBit,
  kTECAddTextRunHeuristicsMask  = 1L << kTECAddTextRunHeuristicsBit,
  kTECAddFallbackInterruptMask  = 1L << kTECAddFallbackInterruptBit
};
```

Unicode Fallback Sequencing Flag

Specifies options for setting fallback sequencing.

```
enum {
    kUnicodeFallbackSequencingBits = 0
};
```

Unicode Fallback Sequencing Masks

Set or text for Unicode sequencing flag.

```
enum {
    kUnicodeFallbackSequencingMask = 3L << kUnicodeFallbackSequencingBits,
    kUnicodeFallbackInterruptSafeMask = 1L << 2
};
```

Constants

`kUnicodeFallbackSequencingMask`

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeFallbackInterruptSafeMask`

Indicate that the caller's fallback routine doesn't move memory.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

Unicode Matching Flags

Specify matching criteria for Unicode mappings.

```
enum {
    kUnicodeMatchUnicodeBaseBit = 0,
    kUnicodeMatchUnicodeVariantBit = 1,
    kUnicodeMatchUnicodeFormatBit = 2,
    kUnicodeMatchOtherBaseBit = 3,
    kUnicodeMatchOtherVariantBit = 4,
    kUnicodeMatchOtherFormatBit = 5
};
```

Constants

`kUnicodeMatchUnicodeBaseBit`
Excludes mappings that do not match the text encoding base of the `unicodeEncoding` field of the structure `UnicodeMapping` (page 1967).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchUnicodeVariantBit`
Excludes mappings that do not match the text encoding variant of the `unicodeEncoding` field of the specified Unicode mapping structure.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchUnicodeFormatBit`
Excludes mappings that do not match the text encoding format of the `unicodeEncoding` field of the specified Unicode mapping structure.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchOtherBaseBit`
Excludes mappings that do not match the text encoding base of the `otherEncoding` field of the structure `UnicodeMapping` (page 1967).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchOtherVariantBit`
Excludes mappings that do not match the text encoding variant of the `otherEncoding` field of the specified Unicode mapping structure.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchOtherFormatBit`
Excludes mappings that do not match the text encoding format of the `otherEncoding` field of the specified Unicode mapping structure.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

Unicode Matching Masks

Used to set or test for Unicode matching flags.

```
enum {
    kUnicodeMatchUnicodeBaseMask = 1L << kUnicodeMatchUnicodeBaseBit,
    kUnicodeMatchUnicodeVariantMask = 1L << kUnicodeMatchUnicodeVariantBit,
    kUnicodeMatchUnicodeFormatMask = 1L << kUnicodeMatchUnicodeFormatBit,
    kUnicodeMatchOtherBaseMask = 1L << kUnicodeMatchOtherBaseBit,
    kUnicodeMatchOtherVariantMask = 1L << kUnicodeMatchOtherVariantBit,
    kUnicodeMatchOtherFormatMask = 1L << kUnicodeMatchOtherFormatBit
};
```

Constants

`kUnicodeMatchUnicodeBaseMask`
If set, excludes mappings that do not match the text encoding base of the `unicodeEncoding` field of the structure `UnicodeMapping` (page 1967). If not set, the function ignores the text encoding base of that field.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchUnicodeVariantMask`
If set, excludes mappings that do not match the text encoding variant of the `unicodeEncoding` field of the specified Unicode mapping structure. If not set, the function ignores the text encoding variant of that field.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchUnicodeFormatMask`
If set, excludes mappings that do not match the text encoding format of the `unicodeEncoding` field of the specified Unicode mapping structure. If not set, the function ignores the text encoding format of that field.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchOtherBaseMask`
If set, excludes mappings that do not match the text encoding base of the `otherEncoding` field of the structure `UnicodeMapping` (page 1967). If not set, the function ignores the text encoding base of that field.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchOtherVariantMask`
If set, excludes mappings that do not match the text encoding variant of the `otherEncoding` field of the specified Unicode mapping structure. If not set, the function ignores the text encoding variant of that field.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchOtherFormatMask`
If set, excludes mappings that do not match the text encoding format of the `otherEncoding` field of the specified Unicode mapping structure. If not set, the function ignores the text encoding format of that field.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

Fallback Handler Selectors

Specify a fallback handler for the Unicode Converter to use.

```
enum {
    kUnicodeFallbackDefaultOnly = 0,
    kUnicodeFallbackCustomOnly = 1,
    kUnicodeFallbackDefaultFirst = 2,
    kUnicodeFallbackCustomFirst = 3
};
```

Constants

`kUnicodeFallbackDefaultOnly`
Use the default fallback handler only.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeFallbackCustomOnly`
Use the custom fallback handler only.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeFallbackDefaultFirst`
Use the default fallback handler first, then the custom one.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeFallbackCustomFirst`
Use the custom fallback handler first, then the default one.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

Discussion

Used to specify which fallback handler the Unicode Converter should use. If you use both the custom and default handlers, you can set the order in which they are called. You use these constants to set the `controlFlags` parameter of the `SetFallbackUnicodeToText` (page 1908) and `SetFallbackUnicodeToTextRun` (page 1909) functions.

Encodings and Variants

Base Text Encodings

Specify base text encodings.

```
typedef UInt32 TextEncodingBase;
enum {
    kTextEncodingMacRoman = 0,
    kTextEncodingMacJapanese = 1,
    kTextEncodingMacChineseTrad = 2,
    kTextEncodingMacKorean = 3,
    kTextEncodingMacArabic = 4,
    kTextEncodingMacHebrew = 5,
    kTextEncodingMacGreek = 6,
    kTextEncodingMacCyrillic = 7,
    kTextEncodingMacDevanagari = 9,
    kTextEncodingMacGurmukhi = 10,
    kTextEncodingMacGujarati = 11,
    kTextEncodingMacOriya = 12,
    kTextEncodingMacBengali = 13,
    kTextEncodingMacTamil = 14,
    kTextEncodingMacTelugu = 15,
    kTextEncodingMacKannada = 16,
    kTextEncodingMacMalayalam = 17,
    kTextEncodingMacSinhalese = 18,
    kTextEncodingMacBurmese = 19,
    kTextEncodingMacKhmer = 20,
    kTextEncodingMacThai = 21,
    kTextEncodingMacLaotian = 22,
    kTextEncodingMacGeorgian = 23,
    kTextEncodingMacArmenian = 24,
    kTextEncodingMacChineseSimp = 25,
    kTextEncodingMacTibetan = 26,
    kTextEncodingMacMongolian = 27,
    kTextEncodingMacEthiopic = 28,
    kTextEncodingMacCentralEurRoman = 29,
    kTextEncodingMacVietnamese = 30,
    kTextEncodingMacExtArabic = 31,
    kTextEncodingMacSymbol = 33,
    kTextEncodingMacDingbats = 34,
    kTextEncodingMacTurkish = 35,
    kTextEncodingMacCroatian = 36,
    kTextEncodingMacIcelandic = 37,
    kTextEncodingMacRomanian = 38,
    kTextEncodingMacCeltic = 39,
    kTextEncodingMacGaelic = 40,
    kTextEncodingMacKeyboardGlyphs = 41
};
```

Constants

`kTextEncodingMacRoman`

The encoding for Mac OS Roman.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingMacJapanese`

The encoding for Mac OS Japanese.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingMacChineseTrad`
The encoding for Mac OS traditional Chinese.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacKorean`
The encoding for Mac OS Korean.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacArabic`
The encoding for Mac OS Arabic.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacHebrew`
The encoding for Mac OS Hebrew.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacGreek`
The encoding for Mac OS Greek.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacCyrillic`
The encoding for Mac OS Cyrillic.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacDevanagari`
The encoding for Mac OS Devanagari.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacGurmukhi`
The encoding for Mac OS Gurmukhi.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacGujarati`
The encoding for Mac OS Gujurati.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacOriya`
The encoding for Mac OS Oriya.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacBengali`
The encoding for Mac OS Bengali.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacTamil`
The encoding for Mac OS Tamil.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacTelugu`
The encoding for Mac OS Telugu.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacKannada`
The encoding for Mac OS Kannada.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacMalayalam`
The encoding for Mac OS Malayalam.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacSinhalese`
The encoding for Mac OS Sinhalese.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacBurmese`
The encoding for Mac OS Burmese.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacKhmer`
The encoding for Mac OS Khmer.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacThai`
The encoding for Mac OS Thai.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacLaotian`
The encoding for Mac OS Laotian.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacGeorgian`
The encoding for Mac OS Georgian.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacArmenian`
The encoding for Mac OS Armenian.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacChineseSimp`
The encoding for Mac OS simple Chinese.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacTibetan`
The encoding for Mac OS Tibetan.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacMongolian`
The encoding for Mac OS Mongolian.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacEthiopic`
The encoding for Mac OS Ethiopic.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacCentralEurRoman`
The encoding for Mac OS Central European Roman.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacVietnamese`
The encoding for Mac OS Vietnamese.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacExtArabic`
The encoding for Mac OS ExtArabic.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacSymbol`
This Mac OS encoding uses script code 0, `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacDingbats`
This Mac OS encoding uses script code 0, `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacTurkish`
This Mac OS encoding uses script code 0, `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacCroatian`
This Mac OS encoding uses script code 0, `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacIcelandic`
This Mac OS encoding uses script code 0, `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacRomanian`
This Mac OS encoding uses script code 0, `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacCeltic`
This Mac OS encoding uses script code 0, `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacGaelic`
This Mac OS encoding uses script code 0, `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacKeyboardGlyphs`
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

Discussion
You use a base text encoding data type to specify which text encoding or text encoding scheme you have used to express a given text. The text encoding base value is the primary specification of the source or target encoding. Values 0 through 32 correspond directly to Mac OS script codes. Values 33 through 254 are for other Mac OS encodings that do not have their own script codes, such as the Symbol encoding implemented by the Symbol font. You can also specify a meta-value as a base text encoding, such as `kTextEncodingMacHFS` and `kTextEncodingUnicodeDefault`. A meta-value is mapped to a real value.

The function `GetTextEncodingBase` (page 1899) returns the text encoding base of a text encoding specification.

A base text encoding is defined by the `TextEncodingBase` data type.

Compatibility TextEncodings

Specify text encodings that are provided for backward compatibility.

```
enum {
    kTextEncodingMacTradChinese = kTextEncodingMacChineseTrad,
    kTextEncodingMacRSymbol = 8,
    kTextEncodingMacSimpChinese = kTextEncodingMacChineseSimp,
    kTextEncodingMacGeez = kTextEncodingMacEthiopic,
    kTextEncodingMacEastEurRoman = kTextEncodingMacCentralEurRoman,
    kTextEncodingMacUninterp = 32
};
```

EBCDIC and IBM Host Text Encodings

Specify text encodings used by IBM computers.

```
enum {
    kTextEncodingEBCDIC_US = 0x0C01,
    kTextEncodingEBCDIC_CP037 = 0x0C02
};
```

Constants

`kTextEncodingEBCDIC_US`
Basic EBCDIC-US encoding.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingEBCDIC_CP037`
Code page 037, extended EBCDIC-US Latin1.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

Discussion

EBCDIC (Extended Binary- Coded Decimal Interchange Code) is used by IBM computers to represent characters as numbers.

Encoding Variants for Big-5

Specify variants of Big-5 encoding.

```
enum {
    kBig5_BasicVariant = 0,
    kBig5_StandardVariant = 1,
    kBig5_ETenVariant = 2
};
```

Constants

`kBig5_BasicVariant`
The basic encoding variant.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kBig5_StandardVariant`
The standard variant; 0xC6A1-0xC7FC: kana, Cyrillic, enclosed numerics.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kBig5_ETenVariant`
Adds kana, Cyrillic, radicals, and so forth with high-bytes C6-C8, F9.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Discussion
Big-5 encoding was developed by five companies as a character set standard in Tawain.

Encoding Variants for Mac OS Encodings

Specify variant Mac OS encodings that use script codes other than 0

```
enum {
    kTextEncodingMacFarsi = 0x8C,
    kTextEncodingMacUkrainian = 0x98,
    kTextEncodingMacInuit = 0xEC,
    kTextEncodingMacVT100 = 0xFC
};
```

Constants
`kTextEncodingMacFarsi`
Uses script code 4, `smArabic`. It is similar to Mac Arabic but uses Farsi digits.]

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingMacUkrainian`
Uses script code 7, `smCyrillic`.]

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingMacInuit`
Uses script code 28, `smEthiopic`.]

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingMacVT100`
Uses script code 32, `smUninterp`; VT100/102 font from the common toolbox; Latin-1 characters plus box drawing.]

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacArabic

Specify variants of MacArabic.

```
enum {
    kMacArabicStandardVariant = 0,
    kMacArabicTrueTypeVariant = 1,
    kMacArabicThuluthVariant = 2,
    kMacArabicAlBayanVariant = 3
};
```

Constants

`kMacArabicStandardVariant`
A Mac OS Arabic variant is supported by the Cairo font (the system font for Arabic) and is the encoding supported by the text processing utilities.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacArabicTrueTypeVariant`
A Mac OS Arabic variant used for most of the Arabic TrueType fonts: Baghdad, Geeza, Kufi, Nadeem.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacArabicThuluthVariant`
A Mac OS Arabic variant used for the Arabic PostScript-only fonts: Thuluth and Thuluth bold.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacArabicAlBayanVariant`
A Mac OS Arabic variant used for the Arabic TrueType font Al Bayan.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacCroatian

Specify variants of MacCroation.

```
enum {
    kMacCroatianDefaultVariant = 0,
    kMacCroatianCurrencySignVariant = 1,
    kMacCroatianEuroSignVariant = 2
};
```

Constants

`kMacCroatianDefaultVariant`
This is a meta value that maps to one of the following constants, depending on version of the Mac OS.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacCroatianCurrencySignVariant`
In versions of Mac OS earlier than 8.5, 0xDB is the currency sign.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacCroatianEuroSignVariant`
In Mac OS version 8.5 and later, 0xDB is the Euro sign.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacCyrillic

Specify variants of MacCyrillic.

```
enum {
    kMacCyrillicDefaultVariant = 0,
    kMacCyrillicCurrSignStdVariant = 1,
    kMacCyrillicCurrSignUkrVariant = 2,
    kMacCyrillicEuroSignVariant = 3
};
```

Constants

`kMacCyrillicDefaultVariant`
This is a meta value that maps to one of the following constants, depending on version of the Mac OS.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacCyrillicCurrSignStdVariant`
In Mac OS versions prior to 9.0 (RU, BG), 0xFF = currency sign, 0xA2/0xB6 = CENT / PARTIAL DIFF.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacCyrillicCurrSignUkrVariant`
In Mac OS version 9.0 and later (UA, LangKit), 0xFF = currency sign, 0xA2/0xB6 = GHE with upturn.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacCyrillicEuroSignVariant`
In Mac OS 9.0 and later, 0xFF is Euro sign, 0xA2/0xB6 = GHE with upturn.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacFarsi

Specify variants of MacFarsi.

```
enum {
    kMacFarsiStandardVariant = 0,
    kMacFarsiTrueTypeVariant = 1
};
```

Constants

`kMacFarsiStandardVariant`
This Mac OS Farsi variant is supported by the Tehran font (the system font for Farsi) and is the encoding supported by the text processing utilities.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacFarsiTrueTypeVariant`
This Mac OS Farsi variant is used for most of the Farsi TrueType fonts: Ashfahan, Amir, Kamran, Mashad, NadeemFarsi.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacHebrew

Specify variants of MacHebrew.

```
enum {
    kMacHebrewStandardVariant = 0,
    kMacHebrewFigureSpaceVariant = 1
};
```

Constants

`kMacHebrewStandardVariant`
The standard Mac OS Hebrew variant.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacHebrewFigureSpaceVariant`
The Mac OS Hebrew variant in which 0xD4 represents figure space, not left single quotation mark as in the standard variant.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacIcelandic

Specify variants of MacIcelandic.

```
enum {
    kMacIcelandicStdDefaultVariant = 0,
    kMacIcelandicTTDefaultVariant = 1,
    kMacIcelandicStdCurrSignVariant = 2,
    kMacIcelandicTTCurrSignVariant = 3,
    kMacIcelandicStdEuroSignVariant = 4,
    kMacIcelandicTTEuroSignVariant = 5
};
```

Constants

`kMacIcelandicStdDefaultVariant`
This is a meta value that maps to `kMacIcelandicStdCurrSignVariant` or `kMacIcelandicStdEuroSignVariant`, depending on version of the Mac OS.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacIcelandicTTDefaultVariant`
This is a meta value that maps to `kMacIcelandicTTCurrSignVariant` or `kMacIcelandicTTEuroSignVariant`, depending on version of the Mac OS.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacIcelandicStdCurrSignVariant`
In Mac OS versions prior to 8.5, 0xDB is the currency sign; 0xBB/0xBC are fem./masc. ordinal indicators.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacIcelandicTTCurrSignVariant`
In Mac OS versions prior to 8.5, 0xDB is the currency sign; 0xBB/0xBC are fi/fl ligatures

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacIcelandicStdEuroSignVariant`
In Mac OS version 8.5 and later , 0xDB is the Euro sign ; 0xBB/0xBC are fem./masc. ordinal indicators.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacIcelandicTTEuroSignVariant`
In Mac OS versions earlier than 8.5, 0xDB is the Euro sign; 0xBB/0xBC are fi/fl ligatures.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacJapanese

Specify variants of MacJapanese.

```
enum {
    kMacJapaneseStandardVariant = 0,
    kMacJapaneseStdNoVerticalsVariant = 1,
    kMacJapaneseBasicVariant = 2,
    kMacJapanesePostScriptScrnVariant = 3,
    kMacJapanesePostScriptPrintVariant = 4,
    kMacJapaneseVertAtKuPlusTenVariant = 5
};
```

Constants

`kMacJapaneseStandardVariant`
The standard Mac OS Japanese variant. Shift-JIS with JIS Roman modifications, extra 1-byte characters, 2-byte Apple extensions, and some vertical presentation forms in the range 0xEB40—0xEDFE ("ku plus 84").

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacJapaneseStdNoVerticalsVariant`
An artificial Mac OS Japanese variant for callers who don't want to use separately encoded vertical forms (for example, developers using QuickDraw GX).

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacJapaneseBasicVariant`
An artificial Mac OS Japanese variant without Apple double-byte extensions.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacJapanesePostScriptScrnVariant`
The Mac OS Japanese variant for the screen bitmap version of the Sai Mincho and Chu Gothic fonts.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacJapanesePostScriptPrintVariant`
The Mac OS Japanese variant for PostScript printing versions of the Sai Mincho and Chu Gothic PostScript fonts. This version includes double-byte half-width characters in addition to single-byte half-width characters.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacJapaneseVertAtKuPlusTenVariant`
The Mac OS Japanese variant for the Hon Mincho and Maru Gothic fonts used in the Japanese localized version of System 7.1. It does not include the standard Apple extensions, and encodes vertical forms at a different location.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacRoman

Specify variants of MacRoman.

```
enum {
    kMacRomanStandardVariant = 0,
    kMacIcelandicStandardVariant = 0,
    kMacIcelandicTrueTypeVariant = 1,
    kJapaneseStandardVariant = 0,
    kJapaneseStdNoVerticalsVariant = 1,
    kJapaneseBasicVariant = 2,
    kJapanesePostScriptScrnVariant = 3,
    kJapanesePostScriptPrintVariant = 4,
    kJapaneseVertAtKuPlusTenVariant = 5,
    kHebrewStandardVariant = 0,
    kHebrewFigureSpaceVariant = 1,
    kUnicodeMaxDecomposedVariant = 2,
    kUnicodeNoComposedVariant = 3,
    kJapaneseNoOneByteKanaOption = 0x20,
    kJapaneseUseAsciiBackslashOption = 0x40
};
```

Constants

`kMacRomanStandardVariant`
The standard variant of Mac OS Roman for Mac OS 8.5 and later; 0xDB is the Euro sign.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacIcelandicStandardVariant`
The standard Mac OS Icelandic encoding supported by the bitmap versions of Chicago, Geneva, Monaco, and New York in the Icelandic system. This is also the variant supported by the text processing utilities.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacIcelandicTrueTypeVariant`
The Mac OS Icelandic variant used for the bitmap versions of Courier, Helvetica, Palatino, and Times in the Icelandic system, and for the TrueType versions of Chicago, Geneva, Monaco, New York, Courier, Helvetica, Palatino, and Times.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kJapaneseStandardVariant`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kJapaneseStdNoVerticalsVariant`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kJapaneseBasicVariant`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kJapanesePostScriptScrnVariant`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kJapanesePostScriptPrintVariant`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kJapaneseVertAtKuPlusTenVariant`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kHebrewStandardVariant`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kHebrewFigureSpaceVariant`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeMaxDecomposedVariant`
Replaced by `kUnicodeCanonicalDecompVariant`.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeNoComposedVariant`
Replaced by `kUnicodeCanonicalCompVariant`.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kJapaneseNoOneByteKanaOption`
Replaced by Unicode Converter option `kUnicodeNoHalfwidthCharsBit`.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `TextCommon.h`.

`kJapaneseUseAsciiBackslashOption`
Replaced by Unicode Converter option `kUnicodeForceASCIIRangeBit`.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `TextCommon.h`.

Encoding Variants for MacRoman Related to Currency

Specify variants of MacRoman that are related to currency.

```
enum {
    kMacRomanDefaultVariant = 0,
    kMacRomanCurrencySignVariant = 1,
    kMacRomanEuroSignVariant = 2
};
```

Constants

`kMacRomanDefaultVariant`
This is a meta value that maps to one of the following constants, depending on version of the Mac OS.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacRomanCurrencySignVariant`
In Mac OS versions earlier than 8.5 0xDB is the currency sign; still used for some older fonts even in Mac OS 8.5.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacRomanEuroSignVariant`
In Mac OS version 8.5 and later, 0xDB is the Euro sign.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacRomanian

Specify variants of MacRomanian.

```
enum {
    kMacRomanianDefaultVariant = 0,
    kMacRomanianCurrencySignVariant = 1,
    kMacRomanianEuroSignVariant = 2
};
```

Constants

`kMacRomanianDefaultVariant`
This is a meta value that maps to one of the following constants, depending on version of the Mac OS.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacRomanianCurrencySignVariant`
In Mac OS versions earlier than 8.5, 0xDB is the currency sign.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacRomanianEuroSignVariant`
In Mac OS version 8.5 and later, 0xDB is the Euro sign.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacRomanLatin1

Specify variants of MacRomanLatin1.

```
enum {
    kMacRomanLatin1DefaultVariant = 0,
    kMacRomanLatin1StandardVariant = 2,
    kMacRomanLatin1TurkishVariant = 6,
    kMacRomanLatin1CroatianVariant = 8,
    kMacRomanLatin1IcelandicVariant = 11,
    kMacRomanLatin1RomanianVariant = 14
};
```

Constants

`kMacRomanLatin1DefaultVariant`
This is a meta value that maps to one of the following constants, depending on version of the Mac OS.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacRomanLatin1StandardVariant`
Permuted MacRoman, Euro sign variant.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacRomanLatin1TurkishVariant`
Permuted MacTurkish.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacRomanLatin1CroatianVariant`
Permuted MacCroatian, Euro sign variant.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacRomanLatin1IcelandicVariant`
Permuted MacIcelandic, standard Euro sign variant.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacRomanLatin1RomanianVariant`
Permuted MacRomanian, Euro sign variant.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacVT100

Specify variants of MacVT100.

```
enum {
    kMacVT100DefaultVariant = 0,
    kMacVT100CurrencySignVariant = 1,
    kMacVT100EuroSignVariant = 2
};
```

Constants

`kMacVT100DefaultVariant`

This is a meta value that maps to one of the following constants, depending on version of the Mac OS.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacVT100CurrencySignVariant`

In Mac OS versions earlier than 8.5, 0xDB is the currency sign.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacVT100EuroSignVariant`

In Mac OS version 8.5 and later, 0xDB is the Euro sign.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for Unicode

Specify variants of Unicode.

```
enum {
    kUnicodeNoSubset = 0,
    kUnicodeCanonicalDecompVariant = 2,
    kUnicodeCanonicalCompVariant = 3,
    kUnicodeHFSPlusDecompVariant = 8,
    kUnicodeHFSPlusCompVariant = 9
};
```

Constants

`kUnicodeNoSubset`

The standard Unicode encoded character set in which the full set of Unicode characters are supported.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeCanonicalDecompVariant`

A variant of Unicode using maximal decomposition with characters in canonical order. This variant does not include most characters which have a canonical decomposition, such as single characters for accented Latin letters or single characters for Korean Hangul syllables (however, this restriction is relaxed for symbol characters in the range U+2000 to U+2FFF). In TEC Manager 1.3, the Unicode Converter supports this variant for converting to and from Mac OS encodings.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeCanonicalCompVariant`
This is the normal canonical composition according to Unicode 3.2 rules.

Available in Mac OS X v10.2 and later.

Declared in `TextCommon.h`.

`kUnicodeHFSPlusDecompVariant`
Specifies canonical decomposition according to Unicode 3.2 rules, with HFS+ exclusions ("HFS+ decomposition 3.2"). That is, it doesn't decompose in 2000-2FFF, F900-FAFF, 2F800-2FAFF. You can use ths option when converting HFS file names.

Available in Mac OS X v10.2 and later.

Declared in `TextCommon.h`.

`kUnicodeHFSPlusCompVariant`
Specifies canonical composition according to Unicode 3.2 rules, but using the HFS+ decomposition exclusions. You can use ths option when converting HFS file names. You should use this form when you want to obtain a composed form that can be converted to and from the decomposed form specified by `kUnicodeHFSPlusDecompVariant`. This is the recommended way to request decompositions with HFS+ exclusions, instead of using `mappingVersion = kUnicodeUseHFSPlusMapping`.

Available in Mac OS X v10.2 and later.

Declared in `TextCommon.h`.

EUC Text Encodings

Specify Extendec Unix Code text encodings.

```
enum {
    kTextEncodingEUC_JP = 0x0920,
    kTextEncodingEUC_CN = 0x0930,
    kTextEncodingEUC_TW = 0x0931,
    kTextEncodingEUC_KR = 0x0940
};
```

Constants

`kTextEncodingEUC_JP`
ISO 646,1-byte katakana,JIS 208 ,JIS 212.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingEUC_CN`
ISO 646, GB 2312-80.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingEUC_TW`
ISO 646, CNS 11643-1992 Planes 1-16.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingEUC_KR`
ISO 646, KS C 5601-1987.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

HFS Text Encoding

Specifies a Mac OS HFS text encoding.

```
enum {
    kTextEncodingMacHFS = 0xFF
};
```

Constants

`kTextEncodingMacHFS`
This is a metavalue for a special Mac OS encoding.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

ISO 2022 Text Encodings

Specify text encodings for ISO 2002.

```
enum {
    kTextEncodingISO_2022_JP = 0x0820,
    kTextEncodingISO_2022_JP_2 = 0x0821,
    kTextEncodingISO_2022_JP_1 = 0x0822,
    kTextEncodingISO_2022_JP_3 = 0x0823,
    kTextEncodingISO_2022_CN = 0x0830,
    kTextEncodingISO_2022_CN_EXT = 0x0831,
    kTextEncodingISO_2022_KR = 0x0840
};
```

Constants

`kTextEncodingISO_2022_JP`
See RFC 1468.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISO_2022_JP_2`
See RFC 1554.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISO_2022_JP_1`
See RFC 2237.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISO_2022_JP_3`
JIS X0213

Available in Mac OS X v10.1 and later.

Declared in `TextCommon.h`.

`kTextEncodingISO_2022_CN`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISO_2022_CN_EXT`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISO_2022_KR`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

ISO 8-bit and 7-bit Text Encodings

Specify text encodings for ISO 8-bit and 7-bit.

```
enum {
    kTextEncodingISOLatin1 = 0x0201,
    kTextEncodingISOLatin2 = 0x0202,
    kTextEncodingISOLatin3 = 0x0203,
    kTextEncodingISOLatin4 = 0x0204,
    kTextEncodingISOLatinCyrillic = 0x0205,
    kTextEncodingISOLatinArabic = 0x0206,
    kTextEncodingISOLatinGreek = 0x0207,
    kTextEncodingISOLatinHebrew = 0x0208,
    kTextEncodingISOLatin5 = 0x0209,
    kTextEncodingISOLatin6 = 0x020A,
    kTextEncodingISOLatin7 = 0x020D,
    kTextEncodingISOLatin8 = 0x020E,
    kTextEncodingISOLatin9 = 0x020F
};
```

Constants

`kTextEncodingISOLatin1`
ISO 8859-1.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatin2`
ISO 8859-2.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatin3`
ISO 8859-3.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatin4`
ISO 8859-4.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatinCyrillic`
ISO 8859-5.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatinArabic`
ISO 8859-6; equivalent to ASMO 708 and DOS CP 708.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatinGreek`
ISO 8859-7.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatinHebrew`
ISO 8859-8.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatin5`
ISO 8859-9.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatin6`
ISO 8859-10.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatin7`
ISO 8859-13; Baltic Rim

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatin8`
ISO 8859-14; Celtic

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatin9`
ISO 8859-15, 8859-1; changed for Euro & CP1252 letters

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Mac Unicode Text Encoding

Specifies a script code that should be handled as a special Mac OS script code.

```
enum {
    kTextEncodingMacUnicode = 0x7E
};
```

Constants

`kTextEncodingMacUnicode`

Beginning with Mac OS 8.5, the set of Mac OS script codes has been extended for some Mac OS components to include Unicode. Some of these components have only 7 bits available for script code, so `kTextEncodingUnicodeDefault` cannot be used to indicate Unicode. Instead, `kTextEncodingMacUnicode` is used as a meta-value to indicate that Unicode handles the script code a special Mac OS script code. The Text Encoding Converter handles this value similar to the way it handles the constant `kTextEncodingUnicodeDefault`.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Miscellaneous Text Encoding Standards

Specify miscellaneous text encodings.

```
enum {
    kTextEncodingShiftJIS = 0x0A01,
    kTextEncodingKOI8_R = 0x0A02,
    kTextEncodingBig5 = 0x0A03,
    kTextEncodingMacRomanLatin1 = 0x0A04,
    kTextEncodingHZ_GB_2312 = 0x0A05,
    kTextEncodingBig5_HKSCS_1999 = 0x0A06
};
```

Constants

`kTextEncodingShiftJIS`

Plain Shift-JIS.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingKOI8_R`

Russian Internet standard.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingBig5`

Big-5 encoding.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingMacRomanLatin1`

Mac OS Roman permuted to align with 8859-1.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingHZ_GB_2312`
See RFC 1842; for Chinese mail and news.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingBig5_HKSCS_1999`
Available in Mac OS X v10.1 and later.

Declared in `TextCommon.h`.

MS-DOS and Windows Text Encodings

Specify text encodings for MS-DOS and Windows.

```
enum {
    kTextEncodingDOSLatinUS = 0x0400,
    kTextEncodingDOSGreek = 0x0405,
    kTextEncodingDOSBalticRim = 0x0406,
    kTextEncodingDOSLatin1 = 0x0410,
    kTextEncodingDOSGreek1 = 0x0411,
    kTextEncodingDOSLatin2 = 0x0412,
    kTextEncodingDOSCyrillic = 0x0413,
    kTextEncodingDOSTurkish = 0x0414,
    kTextEncodingDOSPortuguese = 0x0415,
    kTextEncodingDOSIcelandic = 0x0416,
    kTextEncodingDOSHebrew = 0x0417,
    kTextEncodingDOSCanadianFrench = 0x0418,
    kTextEncodingDOSArabic = 0x0419,
    kTextEncodingDOSNordic = 0x041A,
    kTextEncodingDOSRussian = 0x041B,
    kTextEncodingDOSGreek2 = 0x041C,
    kTextEncodingDOSThai = 0x041D,
    kTextEncodingDOSJapanese = 0x0420,
    kTextEncodingDOSChineseSimplif = 0x0421,
    kTextEncodingDOSKorean = 0x0422,
    kTextEncodingDOSChineseTrad = 0x0423,
    kTextEncodingWindowsLatin1 = 0x0500,
    kTextEncodingWindowsANSI = 0x0500,
    kTextEncodingWindowsLatin2 = 0x0501,
    kTextEncodingWindowsCyrillic = 0x0502,
    kTextEncodingWindowsGreek = 0x0503,
    kTextEncodingWindowsLatin5 = 0x0504,
    kTextEncodingWindowsHebrew = 0x0505,
    kTextEncodingWindowsArabic = 0x0506,
    kTextEncodingWindowsBalticRim = 0x0507,
    kTextEncodingWindowsVietnamese = 0x0508,
    kTextEncodingWindowsKoreanJohab = 0x0510
};
```

Constants

`kTextEncodingDOSLatinUS`
Code page 437.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingDOSGreek`
Code page 737, formerly 437G.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSBalticRim`
Code page 775.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSLatin1`
Code page 860. “multilingual.”
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSGreek1`
Code page 851.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSLatin2`
Code page 852, Slavic.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSCyrillic`
Code page 855, IBM Cyrillic.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSTurkish`
Code page 857, IBM Turkish.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSPortuguese`
Code page 860.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSIcelandic`
Code page 861.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSHebrew`
Code page 862.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSCanadianFrench`
Code page 863.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSArabic`
Code page 864.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSNordic`
Cde page 865.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSRussian`
Code page 866.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSGreek2`
Code page 869, IBM Modern Green.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSThai`
Code page 874, also for Windows.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSJapanese`
Code page 932, also for Windows
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSChineseSimplif`
Code page 936, also for Windows.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSKorean`
Code page 949, also for Windows; unified Hangul.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSChineseTrad`
Code page 950, also for Windows.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsLatin1`
Code page 1252.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsANSI`
Code page 1252 (alternate name).
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsLatin2`
Code page 1250, Central Europe.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsCyrillic`
Code page 1251, Slavic Cyrillic.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsGreek`
Code page 1253.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsLatin5`
Code page 1254, Turkish.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsHebrew`
Code page 1255.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsArabic`
Code page 1256.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsBalticRim`
Code page 1257.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsVietnamese`
Code page 1258.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsKoreanJohab`
Code page 1361, for Window NT.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

National Standard Text Encodings

Specify text encodings for various national standards.

```
enum {
    kTextEncodingUS_ASCII = 0x0600,
    kTextEncodingJIS_X0201_76 = 0x0620,
    kTextEncodingJIS_X0208_83 = 0x0621,
    kTextEncodingJIS_X0208_90 = 0x0622,
    kTextEncodingJIS_X0212_90 = 0x0623,
    kTextEncodingJIS_C6226_78 = 0x0624,
    kTextEncodingShiftJIS_X0213_00 = 0x0628,
    kTextEncodingGB_2312_80 = 0x0630,
    kTextEncodingGBK_95 = 0x0631,
    kTextEncodingGB_18030_2000 = 0x0632,
    kTextEncodingKSC_5601_87 = 0x0640,
    kTextEncodingKSC_5601_92_Johab = 0x0641,
    kTextEncodingCNS_11643_92_P1 = 0x0651,
    kTextEncodingCNS_11643_92_P2 = 0x0652,
    kTextEncodingCNS_11643_92_P3 = 0x0653
};
```

Constants

`kTextEncodingUS_ASCII`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingJIS_X0201_76`
JIS Roman and 1-byte katakana (halfwidth).

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingJIS_X0208_83`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingJIS_X0208_90`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingJIS_X0212_90`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingJIS_C6226_78`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingShiftJIS_X0213_00`
Shift-JIS format encoding of JIS X0213 planes 1 and 2

Available in Mac OS X v10.1 and later.

Declared in `TextCommon.h`.

`kTextEncodingGB_2312_80`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingGBK_95`
Annex to GB13000-93, for Windows 95; EUC-CN extended.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingGB_18030_2000`
Available in Mac OS X v10.1 and later.

Declared in `TextCommon.h`.

`kTextEncodingKSC_5601_87`
This is the same as KSC 5601-92 without Johab annex.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingKSC_5601_92_Johab`
KSC 5601-92 Johab annex.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingCNS_11643_92_P1`
CNS 11643-1992 plane 1.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingCNS_11643_92_P2`
CNS 11643-1992 plane 2.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingCNS_11643_92_P3`
CNS 11643-1992 plane 3 (11643-1986 plane 14).

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

NextStep Platform Encodings

Specify text encodings for the NextStep platform.

```
enum {
    kTextEncodingNextStepLatin = 0x0B01,
    kTextEncodingNextStepJapanese = 0x0B02
};
```

Special Text Encoding Values

Specify special cases of text encodings.

```
enum {
    kTextEncodingMultiRun = 0x0FFF,
    kTextEncodingUnknown = 0xFFFF
};
```

Constants

`kTextEncodingMultiRun`

This is a special value for multiple encoded text, external run information.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingUnknown`

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Text Encoding Formats

Specify a text encoding format.

```
typedef UInt32 TextEncodingFormat;
enum {
    kTextEncodingDefaultFormat = 0,
    kUnicode16BitFormat = 0,
    kUnicodeUTF7Format = 1,
    kUnicodeUTF8Format = 2,
    kUnicode32BitFormat = 3
};
```

Constants

`kTextEncodingDefaultFormat`

The standard default format for any base encoding.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicode16BitFormat`

The 16-bit character encoding format specified by the Unicode standard, equivalent to the UCS-2 format for ISO 10646. This includes support for the UTF-16 method of including non-BMP characters in a stream of 16-bit values.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeUTF7Format`
: The Unicode transformation format in which characters encodings are represented by a sequence of 7-bit values. This format cannot be handled by the Unicode Converter, only by the Text Encoding Converter.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeUTF8Format`
: The Unicode transformation format in which characters are represented by a sequence of 8-bit values.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicode32BitFormat`
: The UCS-4 32-bit format defined for ISO 10646. This format is not currently supported.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Discussion
A text encoding format specifies a way of formatting or algorithmically transforming a particular base encoding. For example, the UTF-7 format is the Unicode standard formatted for transmission through channels that can handle only 7-bit values. Other text encoding formats for Unicode include UTF-8 and 16-bit or 32-bit formats. These transformations are not viewed as different base encodings. Rather, they are different formats for representing the same base encoding.

Similar to text encoding variant values, text encoding format values are specific to a particular text encoding base value or to a small set of text encoding base values. A text encoding format is defined by the `TextEncodingFormat` data type.

The function `GetTextEncodingFormat` (page 1899) returns the text encoding format of a text encoding specification.

Text Encoding Name Selectors

Specify the part of an encoding name you want to obtain.

```
typedef UInt32 TextEncodingNameSelector;
enum {
    kTextEncodingFullName = 0,
    kTextEncodingBaseName = 1,
    kTextEncodingVariantName = 2,
    kTextEncodingFormatName = 3
};
```

Constants
`kTextEncodingFullName`
: Requests the full name of the text encoding.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingBaseName`
: Requests the name of the base encoding.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingVariantName`
Requests the name of the encoding variant, if available.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingFormatName`
Requests the name of the encoding format, if available.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Discussion
You use a selector for the `GetTextEncodingName` function to indicate which part of an encoding name you want to determine. The text encoding name selector is defined by the `TextEncodingNameSelector` data type.

Text Encoding Variants

Specify minor variants of a base encoding or group of base encodings.

```
enum {
    kTextEncodingDefaultVariant = 0
};
```

Constants
`kTextEncodingDefaultVariant`
The standard default variant for any base encoding.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Discussion
This enumeration defines constants for the default variant of any base text encoding and for variants of the Mac OS Japanese, Mac OS Arabic, Mac OS Farsi, Mac OS Hebrew, and Unicode base encodings.

A text encoding variant specifies one among possibly several minor variants of a particular base encoding or group of base encodings. Text encoding variants are often used to support special cases such as the following:

- Differences among fonts that are all intended to support the same encoding. For example, different fonts associated with the MacJapanese and MacArabic encodings support slightly different encoding variants. These fonts would typically coexist on the same system without the user being aware of any differences.
- Artificial variants created by excluding some of the characters in an encoding. For example, the MacJapanese encoding includes separately-encoded vertical forms for some characters. In some contexts (such as with QuickDraw GX), it may be desirable to exclude these.
- Different mappings of a particular character or group of characters for different usages.

For a given text encoding base or small set of related text encoding base values, there may be an enumeration of `TextEncodingVariant` values, which always begins with 0, the default variant. In addition, for a possibly larger set of related text encoding base values, there may be bit masks that can be used independently to designate additional artificial variants. For example, there is an enumeration of six variants for the Mac OS

Japanese encoding. In addition, there are bit masks that can also be used as part of the variant for any Japanese encoding to exclude 1-byte kana or to control the mapping of the reverse solidus (backslash) character.

Languages that are dissimilar but use similar character sets are generally not designated as variants of the same base encoding (for example, MacIcelandic and MacTurkish both use a slight modification of the MacRoman character set, but they are considered separate base encodings).

When you create a new text encoding, you can specify an explicit variant of a base encoding or you can specify the default variant of that base. A text encoding variant is defined by the `TextEncodingVariant` data type. The function `GetTextEncodingVariant` (page 1901) returns the text encoding variant of a text encoding specification.

Unicode and ISO UCS Text Encodings

Specify Unicode and IOS UCS text encodings.

```
enum {
    kTextEncodingUnicodeDefault = 0x0100,
    kTextEncodingUnicodeV1_1 = 0x0101,
    kTextEncodingISO10646_1993 = 0x0101,
    kTextEncodingUnicodeV2_0 = 0x0103,
    kTextEncodingUnicodeV2_1 = 0x0103,
    kTextEncodingUnicodeV3_0 = 0x0104,
    kTextEncodingUnicodeV3_1 = 0x0105,
    kTextEncodingUnicodeV3_2 = 0x0106
};
```

Constants

`kTextEncodingUnicodeDefault`
This is a meta value that takes on one of the following values, depending on the system.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingUnicodeV1_1`
This is a Unicode encoding.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingISO10646_1993`
This ISO UCS encoding has code points identical to Unicode 1.1.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingUnicodeV2_0`
This is the new location for Korean Hangul.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingUnicodeV2_1`
For the Text Encoding Converter, Unicode 2.0 is equivalent to Unicode 2.1.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingUnicodeV3_0`
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingUnicodeV3_1`
Adds characters requiring surrogate pairs in UTF-16
Available in Mac OS X v10.1 and later.
Declared in `TextCommon.h`.

`kTextEncodingUnicodeV3_2`
Available in Mac OS X v10.1 and later.
Declared in `TextCommon.h`.

Unsupported Unicode Variants

Represent Unicode variants that are not yet supported or fully defined.

```
enum {
    kUnicodeNoCompatibilityVariant = 1,
    kUnicodeNoCorporateVariant = 4
};
```

Assorted Constants

Bidirectional Character Values

Specify bidirectional character properties.

```
enum {
    kUCBidiCatNotApplicable = 0,
    kUCBidiCatLeftRight = 1,
    kUCBidiCatRightLeft = 2,
    kUCBidiCatEuroNumber = 3,
    kUCBidiCatEuroNumberSeparator = 4,
    kUCBidiCatEuroNumberTerminator = 5,
    kUCBidiCatArabicNumber = 6,
    kUCBidiCatCommonNumberSeparator = 7,
    kUCBidiCatBlockSeparator = 8,
    kUCBidiCatSegmentSeparator = 9,
    kUCBidiCatWhitespace = 10,
    kUCBidiCatOtherNeutral = 11,
    kUCBidiCatRightLeftArabic = 12,
    kUCBidiCatLeftRightEmbedding = 13,
    kUCBidiCatRightLeftEmbedding = 14,
    kUCBidiCatLeftRightOverride = 15,
    kUCBidiCatRightLeftOverride = 16,
    kUCBidiCatPopDirectionalFormat = 17,
    kUCBidiCatNonSpacingMark = 18,
    kUCBidiCatBoundaryNeutral = 19
};
```

Constants

`kUCBidiCatNotApplicable`
Unassigned.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatLeftRight`
Strong types: L left-to-right.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatRightLeft`
Strong types: R right-to-left.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatEuroNumber`
Weak types: EN European number.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatEuroNumberSeparator`
Weak types: ES European number separator.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatEuroNumberTerminator`
Weak types: ET European number terminator.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatArabicNumber`
Weak types: AN Arabic number.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatCommonNumberSeparator`
Weak types: CS common number separator.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatBlockSeparator`
Separators: B paragraph separator (was block separator).

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatSegmentSeparator`
Separators: S segment separator.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatWhitespace`
Neutrals: WS whitespace.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatOtherNeutral`
Neutrals: ON other neutrals (unassigned codes could use this).

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatRightLeftArabic`
Unicode 3.0; AL right-to-left Arabic (was Arabic letter).

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatLeftRightEmbedding`
Unicode 3.0; LRE eft-to-right embedding.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatRightLeftEmbedding`
Unicode 3.0; RLE right-to-left embedding.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatLeftRightOverride`
Unicode 3.0; LRO left-to-right override.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatRightLeftOverride`
Unicode 3.0; RLO right-to-left override.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatPopDirectionalFormat`
Unicode 3.0; PDF pop directional Format.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatNonSpacingMark`
Unicode 3.0; NSM non-spacing mark.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatBoundaryNeutral`
Unicode 3.0; BN boundary neutral.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Discussion
These values are requested by `kUCCharPropTypeBidiCategory`.

Common and Special Unicode Values

Specify sommon and special Unicode code values.

```
enum {
    kUnicodeByteOrderMark = 0xFEFF,
    kUnicodeObjectReplacement = 0xFFFC,
    kUnicodeReplacementChar = 0xFFFD,
    kUnicodeSwappedByteOrderMark = 0xFFFE,
    kUnicodeNotAChar = 0xFFFF
};
```

Constants
`kUnicodeByteOrderMark`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeObjectReplacement`
A placeholder for a non-text object.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeReplacementChar`
Unicode replacement for an input character that cannot be converted.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeSwappedByteOrderMark`
Not a Unicode character; byte-swapped version of FEFF.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeNotAChar`
Not a Unicode character; may be used as a terminator.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

TEC Plugin Dispatch Table Versions

Specify a version for a TEC plug-in dispatch table.

```
enum {
    kTECPluginDispatchTableVersion1 = 0x00010000,
    kTECPluginDispatchTableVersion1_1 = 0x00010001,
    kTECPluginDispatchTableVersion1_2 = 0x00010002,
    kTECPluginDispatchTableCurrentVersion = kTECPluginDispatchTableVersion1_2
};
```

Constants

`kTECPluginDispatchTableVersion1`
Specifies versions 1.0 through 1.0.3.

Available in Mac OS X v10.0 and later.

Declared in `TextEncodingPlugin.h`.

`kTECPluginDispatchTableVersion1_1`
Specifies version 1.1.

Available in Mac OS X v10.0 and later.

Declared in `TextEncodingPlugin.h`.

`kTECPluginDispatchTableVersion1_2`
Specifies version 1.2.

Available in Mac OS X v10.0 and later.

Declared in `TextEncodingPlugin.h`.

`kTECPluginDispatchTableCurrentVersion`
A meta value that specifies the current version.

Available in Mac OS X v10.0 and later.

Declared in `TextEncodingPlugin.h`.

TEC Plug-in Signatures

Specify a TEC plug-in signature.

```
enum {
    kTECSignature = 'encv',
    kTECUnicodePluginSignature = 'puni',
    kTECJapanesePluginSignature = 'pjpn',
    kTECChinesePluginSignature = 'pzho',
    kTECKoreanPluginSignature = 'pkor'
};
```

Unicode Character Property Types

Specify property types for a Unicode charater.

```
typedef SInt32 UCCharPropertyType;
enum {
    kUCCharPropTypeGenlCategory = 1,
    kUCCharPropTypeCombiningClass = 2,
    kUCCharPropTypeBidiCategory = 3
};
```

Constants

`kUCCharPropTypeGenlCategory`
Requests enumeration value.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kUCCharPropTypeCombiningClass`
Requests numeric value 0 to 255.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kUCCharPropTypeBidiCategory`
Requests enumeration value.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

Unicode Character Property Values

Specify a propery value for a Unicode character.

```
typedef UInt32 UCCharPropertyValue;
enum {
    kUCGenlCatOtherNotAssigned = 0,
    kUCGenlCatOtherControl = 1,
    kUCGenlCatOtherFormat = 2,
    kUCGenlCatOtherSurrogate = 3,
    kUCGenlCatOtherPrivateUse = 4,
    kUCGenlCatMarkNonSpacing = 5,
    kUCGenlCatMarkSpacingCombining = 6,
    kUCGenlCatMarkEnclosing = 7,
    kUCGenlCatNumberDecimalDigit = 8,
    kUCGenlCatNumberLetter = 9,
    kUCGenlCatNumberOther = 10,
    kUCGenlCatSeparatorSpace = 11,
    kUCGenlCatSeparatorLine = 12,
    kUCGenlCatSeparatorParagraph = 13,
    kUCGenlCatLetterUppercase = 14,
    kUCGenlCatLetterLowercase = 15,
    kUCGenlCatLetterTitlecase = 16,
    kUCGenlCatLetterModifier = 17,
    kUCGenlCatLetterOther = 18,
    kUCGenlCatPunctConnector = 20,
    kUCGenlCatPunctDash = 21,
    kUCGenlCatPunctOpen = 22,
    kUCGenlCatPunctClose = 23,
    kUCGenlCatPunctInitialQuote = 24,
    kUCGenlCatPunctFinalQuote = 25,
    kUCGenlCatPunctOther = 26,
    kUCGenlCatSymbolMath = 28,
    kUCGenlCatSymbolCurrency = 29,
    kUCGenlCatSymbolModifier = 30,
    kUCGenlCatSymbolOther = 31
};
```

Constants

`kUCGenlCatOtherNotAssigned`
Cn other; not assigned.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatOtherControl`
Cc other; control.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatOtherFormat`
Cf other; format.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatOtherSurrogate`
Cs other; surrogate.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatOtherPrivateUse`
Co other; private use.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatMarkNonSpacing`
Mn mark; non-spacing.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatMarkSpacingCombining`
Mc mark; spacing combining.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatMarkEnclosing`
Me mark; enclosing.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatNumberDecimalDigit`
Nd number; decimal digit.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatNumberLetter`
Nl number; letter.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatNumberOther`
No number; other.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatSeparatorSpace`
Zs separator; space.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatSeparatorLine`
Zl separator; Line.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatSeparatorParagraph`
Zp separator; paragraph.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatLetterUppercase`
Lu Letter; uppercase.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatLetterLowercase`
Ll Letter; lowercase.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatLetterTitlecase`
Lt Letter; titlecase.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatLetterModifier`
Lm Letter; modifier.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatLetterOther`
Lo Letter; other.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatPunctConnector`
Pc punctuation; connector.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatPunctDash`
Pd punctuation; dash.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatPunctOpen`
Ps punctuation; open.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatPunctClose`
Pe punctuation; close.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatPunctInitialQuote`
Pi punctuation; initial quote.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatPunctFinalQuote`
Pf punctuation; final quote.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatPunctOther`
Po punctuation; other.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatSymbolMath`
Sm symbol; math.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatSymbolCurrency`
Sc symbol; currency.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatSymbolModifier`
Sk symbol; modifier.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatSymbolOther`
So symbol; other.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Unicode Mapping Versions

Specify a Unicode mapping version.

```
typedef SInt32 UnicodeMapVersion;
enum {
    kUnicodeUseLatestMapping = -1,
    kUnicodeUseHFSPlusMapping = 4
};
```

Constants

`kUnicodeUseLatestMapping`
Instead of explicitly specifying the mapping version of the Unicode mapping table to be used for conversion of a text string, you can use this constant to specify that the latest version be used.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeUseHFSPlusMapping`
Indicates the mapping version used by HFS Plus to convert filenames between Mac OS encodings and Unicode. Only one constant is defined so far for a specific mapping version.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

Discussion
When performing conversions, you specify the version of the Unicode mapping table to be used for the conversion. You provide the version number in the mapping version field of the structure `UnicodeMapping` (page 1967) that is passed to a function. A Unicode mapping version is defined by the `UnicodeMapVersion` data type.

Unwanted Data Constants

Specify data you don't care about receiving.

```
enum {
    kTextScriptDontCare = -128,
    kTextLanguageDontCare = -128,
    kTextRegionDontCare = -128
};
```

Constants
`kTextScriptDontCare`
Indicates that the code is not provided for the derivation.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextLanguageDontCare`
Indicates that language code is not provided for the derivation.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextRegionDontCare`
The region code is not provided for the derivation.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Discussion
For backward compatibility with earlier releases of the Mac OS, the Text Encoding Conversion Manager provides the functions `UpgradeScriptInfoToTextEncoding` (page 1937) and `RevertTextEncodingToScriptInfo` (page 1907) that you can use to derive Script Manager values from a text encoding or vice versa.

When using these functions, you can specify a Script Manager language code, script code, and/or font values to derive a text encoding. These three constants are defined to allow you to identify any part of the derivation you don't care about. When reverting from a text encoding to Script Manager values, the Unicode Converter returns these constants for a corresponding value it does not derive: `kTextLanguageDontCare`, `kTextScriptDontCare`, and `kTextRegionDontCare`.

Result Codes

The most common result codes returned by Text Encoding Conversion Manager are listed below.

Result Code	Value	Description
kTextUnsupportedEncodingErr	-8738	The encoding or mapping is not supported for this function by the current set of tables or plug-ins. Available in Mac OS X v10.0 and later.
kTextMalformedInputErr	-8739	The text input contains a sequence that is not legal in the specified encoding, such as a DBCS high byte followed by an invalid low byte (0x8120 in Shift-JIS). Available in Mac OS X v10.0 and later.
kTextUndefinedElementErr	-8740	The text input contains a code point that is undefined in the specified encoding. The function did not completely convert the input string. You can resume conversion from a point beyond the offending character, or take some other action. Available in Mac OS X v10.0 and later.
kTECMissingTableErr	-8745	The specified encoding is partially supported, but a specific table required for this function is missing. Available in Mac OS X v10.0 and later.
kTECTableChecksumErr	-8746	A specific table required for this function has a checksum error, indicating that it has become corrupted. Available in Mac OS X v10.0 and later.
kTECTableFormatErr	-8747	The table format is either invalid or it cannot be handled by the current version of the code. The function did not convert the string Available in Mac OS X v10.0 and later.
kTECCorruptConverterErr	-8748	The converter object is invalid. Returned by the Text Encoding Converter functions only. Available in Mac OS X v10.0 and later.
kTECNoConversionPathErr	-8749	The converter supports both the source and target encodings, but cannot convert between them either directly or indirectly. Returned by the Text Encoding Converter functions only. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kTECBufferBelowMinimumSizeErr	-8750	The output text buffer is too small to accommodate the result of processing of the first input text element. No part of the input string was processed. Available in Mac OS X v10.0 and later.
kTECArrayFullErr	-8751	The supplied TextEncodingRun, ScriptCodeRun, or UnicodeMapping array is too small, and the input was not completely converted. Call the function again with another output buffer—or with the same output buffer after copying its contents—to convert the remainder of the string Available in Mac OS X v10.0 and later.
kTECPartialCharErr	-8753	The input text ends in the middle of a multibyte character and conversion stopped. Append the unconverted input from this call to the beginning of the subsequent input text and call the function again. Available in Mac OS X v10.0 and later.
kTECUnmappableElementErr	-8754	An input text element cannot be mapped to the specified output encoding(s) using the specified options. For the Unicode Converter, this error can occur only if kUnicodeUseFallbacksBit control flag is not set. Available in Mac OS X v10.0 and later.
kTECIncompleteElementErr	-8755	The input text ends with a text element that might be incomplete, or contains a text element that is too long for the internal buffers. Available in Mac OS X v10.0 and later.
kTECDirectionErr	-8756	An error, such as a direction stack overflow, occurred in directionality processing. Available in Mac OS X v10.0 and later.
kTECGlobalsUnavailableErr	-8770	Global variables have already been deallocated, premature termination. The function did not convert the string. Available in Mac OS X v10.0 and later.
kTECItemUnavailableErr	-8771	An item (for example, a name) is not available for the specified region (and encoding, if relevant). Available in Mac OS X v10.0 and later.
kTECUsedFallbacksStatus	-8783	The function has completely converted the input string to the specified target using one or more fallbacks. For the Unicode Converter, this status code can only occur if the kUnicodeUseFallbacksBit control flag is set. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kTECNeedFlushStatus	-8784	The application disposed of a converter object by calling TECDisposeConverter, but there is still text contained in internal buffers. Returned by the Text Encoding Converter functions only. Available in Mac OS X v10.0 and later.
kTECOutputBufferFullStatus	-8785	The converter successfully converted part of the input text, but the output buffer was not large enough to accommodate the entire input text after conversion. Convert the remaining text beginning from the position where the conversion stopped. Available in Mac OS X v10.0 and later.

Text Utilities Reference

Framework:	CoreServices/CoreServices.h, Carbon/Carbon.h
Declared in	TextUtils.h StringCompare.h NumberFormatting.h TypeSelect.h

Overview

The Text Utilities provide you with an integrated collection of routines for performing a variety of operations on textual information, ranging from modifying the contents of a string, to sorting strings from different languages, to converting times, dates, and numbers from internal representations to formatted strings and back. These routines work in conjunction with QuickDraw text drawing routines to help you display and modify text in applications that are distributed to an international audience.

The Text Utilities functions are used for numerous text-handling tasks, including

- defining strings–including functions for allocating strings in the heap and for loading strings from resources
- comparing and sorting strings–including functions for testing whether two strings are equal and functions for finding the sorting relationship between two strings
- modifying the contents of strings–including routines for converting the case of characters, stripping diacritical marks, replacing substrings, and truncating strings
- finding breaks and boundaries in text–including routines for finding word and line breaks, and for finding different script runs in a line of text
- converting and formatting date and time strings–including routines that convert numeric and string representations of dates and times into record format, and routines that convert numeric and record representations of dates and times into strings
- converting and formatting numeric strings–including routines that convert string representations of numbers into numeric representations

Carbon supports the majority of Text Utilities. However, Apple recommends that you use the comparison and word breaking utilities supplied by Unicode Utilities instead.

A number of obsolete Text Utilities functions-such as those prefixed with iu or IU-are not supported.

Functions by Task

Comparing Strings for Equality

`EqualString` (page 2038) Deprecated in Mac OS X v10.4
: Compares two Pascal strings for equality, using the comparison rules of the Macintosh file system. (Deprecated. Use `CFStringCompare` instead.)

`IdenticalString` (page 2046) Deprecated in Mac OS X v10.4
: Compares two Pascal strings for equality, making use of the string comparison information from a resource that you specify as a parameter. (Deprecated. Use `CFStringCompare` instead.)

`IdenticalText` (page 2046) Deprecated in Mac OS X v10.4
: Compares two text strings for equality, making use of the string comparison information from a resource that you specify as a parameter. (Deprecated. Use `CFStringCompare` instead.)

Converting Between Integers and Strings

`NumToString` (page 2052) Deprecated in Mac OS X v10.4
: Converts a long integer value into a Pascal string. (Deprecated. Use `CFStringCreateWithFormat` instead.)

`StringToNum` (page 2062) Deprecated in Mac OS X v10.4
: Converts the Pascal string representation of a base-10 number into a long integer value. (Deprecated. Use `CFStringGetIntValue` instead.)

Converting Between Strings and Floating-Point Numbers

`ExtendedToString` (page 2039) Deprecated in Mac OS X v10.4
: Converts an internal floating-point representation of a number into a string that can be presented to the user, using a `NumFormatStringRec` structure to specify how the output number string is formatted (Deprecated. Use `CFNumberFormatterCreateNumberFromString` instead.)

`StringToExtended` (page 2059) Deprecated in Mac OS X v10.4
: Converts a string representation of a number into a floating-point number, using a `NumFormatStringRec` structure to specify how the input number string is formatted. (Deprecated. Use `CFNumberFormatterCreateStringWithNumber` instead.)

Converting Between C and Pascal Strings

`c2pstr` (page 2034) Deprecated in Mac OS X v10.4 Deprecated in Mac OS X v10.4
: Converts a C string to a Pascal string. (Deprecated. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

`C2PStr` (page 2034) Deprecated in Mac OS X v10.4 Deprecated in Mac OS X v10.4 Deprecated in Mac OS X v10.4
: Converts a C string to a Pascal string. (Deprecated. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

`c2pstrcpy` (page 2034) Deprecated in Mac OS X v10.4
Converts a C string to a Pascal string. (Deprecated. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

`CopyCStringToPascal` (page 2037) Deprecated in Mac OS X v10.4
Converts a C string to a Pascal string. (Deprecated. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

`CopyPascalStringToC` (page 2037) Deprecated in Mac OS X v10.4
Converts a Pascal String to a C string. (Deprecated. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

`p2cstr` (page 2053) Deprecated in Mac OS X v10.4 Deprecated in Mac OS X v10.4
Converts a Pascal string to a C string. (Deprecated. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

`P2CStr` (page 2053) Deprecated in Mac OS X v10.4 Deprecated in Mac OS X v10.4 Deprecated in Mac OS X version 10.4
Converts a Pascal string to a C string. (Deprecated. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

`p2cstrcpy` (page 2054) Deprecated in Mac OS X v10.4
Converts a Pascal string to a C string. (Deprecated. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

Defining and Specifying Strings

`GetIndString` (page 2044) Deprecated in Mac OS X v10.4
Loads a string from a string list (`'STR#'`) resource into memory, given the resource ID of the string list and the index of the individual string. (Deprecated. Use `CFBundleCopyLocalizedString` instead.)

`GetString` (page 2045) Deprecated in Mac OS X v10.4
Loads a string from a string (`'STR'`) resource into memory. (Deprecated. Use `CFBundleCopyLocalizedString` instead.)

`NewString` (page 2051) Deprecated in Mac OS X v10.4
Allocates memory in the heap for a string, copies its contents, and produces a handle for the heap version of the string. (Deprecated. Use `CFStringCreateCopy` instead.)

`SetString` (page 2057) Deprecated in Mac OS X v10.4
Changes the contents of a string referenced by a string handle, replacing the previous contents by copying the specified string. (Deprecated. Use `CFStringCreateWithPascalString` and `CFStringReplaceAll`.)

Determining Sorting Order for Strings in Different Languages

`LanguageOrder` (page 2048) Deprecated in Mac OS X v10.4
Determines the order in which strings in two different languages should be sorted. (Deprecated. Use `CFStringCompare` or `UCCompareText` (page 2151) instead.)

`ScriptOrder` (page 2057) Deprecated in Mac OS X v10.4
Determines the order in which strings in two different scripts should be sorted. (Deprecated. Use `CFStringCompare` or `UCCompareText` (page 2151) instead.)

`StringOrder` (page 2058) Deprecated in Mac OS X v10.4
Compares two Pascal strings, taking into account the script system and language for each of the strings. (Deprecated. Use `CFStringCompare` or `UCCompareText` (page 2151) instead.)

`TextOrder` (page 2064) Deprecated in Mac OS X v10.4
Compares two text strings, taking into account the script and language for each of the strings. (Deprecated. Use `CFStringCompare` or `UCCompareText` (page 2151) instead.)

Determining Sorting Order for Strings in the Same Language

`CompareString` (page 2035) Deprecated in Mac OS X v10.4
Compares two Pascal strings, making use of the string comparison information from a resource that you specify as a parameter. (Deprecated. Use `CFStringCompare` or `UCCompareText` (page 2151) instead.)

`CompareText` (page 2036) Deprecated in Mac OS X v10.4
Compares two text strings, making use of the string comparison information from a resource that you specify as a parameter. (Deprecated. Use `CFStringCompare` or `UCCompareText` (page 2151) instead.)

`RelString` (page 2054) Deprecated in Mac OS X v10.4
Compares two Pascal strings using the string comparison rules of the Macintosh file system and returns a value that indicates the sorting order of the first string relative to the second string. (Deprecated. Use `CFStringCompare` or `UCCompareText` (page 2151) instead.)

`relstring` (page 2055) Deprecated in Mac OS X v10.4
Compares two strings. (Deprecated. Use `CFStringCompare` or `UCCompareText` (page 2151) instead.)

Modifying Characters and Diacritical Marks

`LowercaseText` (page 2049) Deprecated in Mac OS X v10.4
Converts any uppercase characters in a text string into their lowercase equivalents. (Deprecated. Use `CFStringLowercase` instead.)

`StripDiacritics` (page 2063) Deprecated in Mac OS X v10.4
Strips any diacritical marks from a text string. (Deprecated. Use `CFStringTransform` instead.)

`UppercaseStripDiacritics` (page 2068) Deprecated in Mac OS X v10.4
Converts any lowercase characters in a text string into their uppercase equivalents and strips any diacritical marks from the text. (Deprecated. Use `CFStringTransform` instead.)

`UppercaseText` (page 2069) Deprecated in Mac OS X v10.4
Converts any lowercase characters in a text string into their uppercase equivalents. (Deprecated. Use `CFStringUppercase` instead.)

`UpperString` (page 2070) Deprecated in Mac OS X v10.4
Converts any lowercase letters in a Pascal string to their uppercase equivalents, using the Macintosh file system rules. (Deprecated. Use `CFStringUppercase` instead.)

`upperstring` (page 2071) Deprecated in Mac OS X v10.4
Converts any lowercase letters in a Pascal string to their uppercase equivalents. (Deprecated. Use `CFStringUppercase` instead.)

Searching for and Replacing Strings

`Munger` (page 2049)
Searches text for a specified string pattern and replaces it with another string.

`ReplaceText` (page 2056) Deprecated in Mac OS X v10.4
Searches text on a character-by-character basis, replacing all instances of a string in that text with another string. (Deprecated. Use `CFStringReplace` instead.)

Using Number Format Specification Strings for International Number Formatting

`FormatRecToString` (page 2043) Deprecated in Mac OS X v10.4
Converts an internal representation of number formatting information into a number format specification string, which can be displayed and modified. (Deprecated. Use `CFNumberFormatterGetFormat` instead.)

`StringToFormatRec` (page 2060) Deprecated in Mac OS X v10.4
Creates a number format specification string structure from a number format specification string that you supply in a Pascal string. (Deprecated. Use `CFNumberFormatterSetFormat` instead.)

Working With Word, Script, and Line Boundaries

`FindScriptRun` (page 2040) Deprecated in Mac OS X v10.4
Finds the next block of subscript text within a script run. (Deprecated. There is no replacement function because this capability is no longer needed in Mac OS X.)

`FindWordBreaks` (page 2041) Deprecated in Mac OS X v10.4
Determines the beginning and ending boundaries of a word in a text string. (Deprecated. Use `UCFindTextBreak` (page 2159) instead.)

Working With Universal Procedure Pointers

`DisposeIndexToStringUPP` (page 2038) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer to an index-to-string callback.

`InvokeIndexToStringUPP` (page 2047) Deprecated in Mac OS X v10.4
Call an index-to-string callback.

`NewIndexToStringUPP` (page 2051) Deprecated in Mac OS X v10.4
Creates a new universal procedure pointer (UPP) to an index-to-string callback.

Working With Type Select Records

`TypeSelectClear` (page 2065) Deprecated in Mac OS X v10.4
Clears the key list and resets the type select record. (Deprecated. Use `UCTypeSelectFlushSelectorData` instead.)

`TypeSelectCompare` (page 2066) Deprecated in Mac OS X v10.4
Compares a text buffer to the keystroke buffer. (Deprecated. Use `UCTypeSelectCompare` instead.)

`TypeSelectFindItem` (page 2066) Deprecated in Mac OS X v10.4
: Finds the closest match between a specified list of characters and the keystrokes stored in the type select record. (Deprecated. Use `UCTypeSelectFindItem` instead.)

`TypeSelectNewKey` (page 2067) Deprecated in Mac OS X v10.4
: Creates a new type select record. (Deprecated. Use `UCTypeSelectCreateSelector` instead.)

Functions

c2pstr

Converts a C string to a Pascal string. (Deprecated in Mac OS X v10.4. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

```
StringPtr c2pstr (
   char *aStr
);
```

Availability
Available in Mac OS X v10.4 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextUtils.h`

C2PStr

Converts a C string to a Pascal string. (Deprecated in Mac OS X v10.4. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

```
StringPtr C2PStr (
    Ptr cString
);
```

Availability
Available in Mac OS X v10.4 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextUtils.h`

c2pstrcpy

Converts a C string to a Pascal string. (Deprecated in Mac OS X v10.4. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

```
void c2pstrcpy (
   Str255 dst,
   const char *src
);
```

Parameters

dst

On output, the Pascal string.

src

The C string you want to convert.

Discussion

This function allows in-place conversion. That is, the `src` and `dst` parameters can point to the same memory location.

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

SoftVDigX

Declared In

`TextUtils.h`

CompareString

Compares two Pascal strings, making use of the string comparison information from a resource that you specify as a parameter. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` or `UCCompareText` (page 2151) instead.)

```
short CompareString (
    ConstStr255Param aStr,
    ConstStr255Param bStr,
    Handle itl2Handle
);
```

Parameters

aStr

One of the Pascal strings to be compared.

bStr

The other Pascal string to be compared.

itl2Handle

The handle to the string-manipulation resource that contains string comparison information. If the value of this parameter is `NULL`, `CompareString` makes use of the resource for the current script. The string-manipulation resource includes functions and tables for modifying string comparison and tables for case conversion and stripping of diacritical marks.

Return Value

Returns –1 if the first string is less than the second string, 0 if the first string is equal to the second string, and 1 if the first string is greater than the second string.

Discussion
This function takes both primary and secondary sorting orders into consideration and returns a value that indicates the sorting order of the first string relative to the second string.

Special Considerations

`CompareString` 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
`StringCompare.h`

CompareText

Compares two text strings, making use of the string comparison information from a resource that you specify as a parameter. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` or `UCCompareText` (page 2151) instead.)

```
short CompareText (
   const void *aPtr,
   const void *bPtr,
   short aLen,
   short bLen,
   Handle itl2Handle
);
```

Parameters

aPtr
A pointer to the first character of the first text string.

bPtr
A pointer to the first character of the second text string.

aLen
The length, in bytes, of the first text string.

bLen
The length, in bytes, of the second text string.

itl2Handle
A handle to a string-manipulation (`'itl2'`) resource that contains string comparison information. If the value of this parameter is `NULL`, `CompareText` makes use of the resource for the current script. The string-manipulation resource includes functions and tables for modifying string comparison and tables for case conversion and stripping of diacritical marks.

Return Value
Returns –1 if the first string is less than the second string, 0 if the first string is equal to the second string, and 1 if the first string is greater than the second string.

Discussion
This function takes both primary and secondary sorting orders into consideration and returns a value that indicates the sorting order of the first string relative to the second string.

Special Considerations

`CompareText` 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

`StringCompare.h`

CopyCStringToPascal

Converts a C string to a Pascal string. (Deprecated in Mac OS X v10.4. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

```
void CopyCStringToPascal (
   const char *src,
   Str255 dst
);
```

Parameters

src

The C string you want to convert.

dst

On output, the Pascal string.

Discussion

This function allows in-place conversion. That is, the `src` and `dst` parameters can point to the same memory location.

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

BSDLLCTest

Declared In

`TextUtils.h`

CopyPascalStringToC

Converts a Pascal String to a C string. (Deprecated in Mac OS X v10.4. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

```
void CopyPascalStringToC (
   ConstStr255Param src,
   char *dst
);
```

Parameters

src

The Pascal string you want to convert.

dst

On output, the C string.

Discussion

This function allows in-place conversion. That is, the `src` and `dst` parameters can point to the same memory location.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextUtils.h`

DisposeIndexToStringUPP

Disposes of a universal procedure pointer to an index-to-string callback. (Deprecated in Mac OS X v10.4.)

```
void DisposeIndexToStringUPP (
   IndexToStringUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion

See the callback `IndexToStringProcPtr` (page 2071) for more information.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TypeSelect.h`

EqualString

Compares two Pascal strings for equality, using the comparison rules of the Macintosh file system. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` instead.)

```
Boolean EqualString (
   ConstStr255Param str1,
   ConstStr255Param str2,
   Boolean caseSensitive,
   Boolean diacSensitive
);
```

Parameters

str1

One of the Pascal strings to be compared.

str2

The other Pascal string to be compared.

caseSensitive

A flag that indicates how to handle case-sensitive information during the comparison. If the value of `caseSens` is `TRUE`, uppercase characters are distinguished from the corresponding lowercase characters. If it is `FALSE`, case information is ignored.

diacSensitive

A flag that indicates how to handle information about diacritical marks during the string comparison. If the value of `diacSens` is `TRUE`, characters with diacritical marks are distinguished from the corresponding characters without diacritical marks during the comparison. If it is `FALSE`, diacritical marks are ignored.

Return Value

`TRUE` if the two strings are equal and `FALSE` if they are not equal. If its value is `TRUE`, `EqualString` distinguishes uppercase characters from the corresponding lowercase characters. If its value is `FALSE`, `EqualString` ignores diacritical marks during the comparison.

Discussion

The comparison is a simple, character-by-character value comparison. This function does not make use of any script or language information (i.e., is not localizable); it assumes the use of a Roman script 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

`StringCompare.h`

ExtendedToString

Converts an internal floating-point representation of a number into a string that can be presented to the user, using a `NumFormatStringRec` structure to specify how the output number string is formatted (Deprecated in Mac OS X v10.4. Use `CFNumberFormatterCreateNumberFromString` instead.)

```
FormatStatus ExtendedToString (
   const extended80 *x,
   const NumFormatString *myCanonical,
   const NumberParts *partsTable,
   Str255 outString
);
```

Parameters

x

A pointer to a floating-point value in 80-bit SANE representation.

myCanonical

A pointer to the internal representation of the formatting information for numbers, as produced by the `StringToFormatRec` function.

partsTable

A pointer to a structure, obtained from the tokens (`'itl4'`) resource, that shows the correspondence between generic number part separators (tokens) and their localized version (for example, a thousand separator is a comma in the United States and a decimal point in France).

outString

On output, contains the number formatted according to the information in `myFormatRec`.

Return Value

A value that denotes the confidence level for the conversion that it performed. The low byte of the `FormatStatus` value is of type `FormatResultType`. Be sure to cast the result of `ExtendedToString` to a type `FormatResultType` before working with it. See the description of the `FormatStatus` data type.

Discussion

`ExtendedToString` creates a string representation of a floating-point number, using the formatting information in the `myFormatRec` parameter (which was created by a previous call to `StringToFormatRec`) to determine how the number should be formatted for output. It uses the number parts table to determine the component parts of the number string.

To obtain a handle to the number parts table from a tokens resource, use the `GetIntlResourceTable` 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

`NumberFormatting.h`

FindScriptRun

Finds the next block of subscript text within a script run. (Deprecated in Mac OS X v10.4. There is no replacement function because this capability is no longer needed in Mac OS X.)

```
ScriptRunStatus FindScriptRun (
   Ptr textPtr,
   long textLen,
   long *lenUsed
);
```

Parameters

textPtr

A pointer to the text string to be analyzed.

textLen

The number of bytes in the text string.

lenUsed

On output, a pointer to the length, in bytes, of the script run that begins with the first character in the string; this length is always greater than or equal to 1, unless the string passed in is of length 0.

Return Value

Identifies the run as either native text, Roman, or one of the defined subscripts of the script system and returns a structure of type `ScriptRunStatus` (page 2078). See the description of the `ScriptRunStatus` data type.

Discussion

The `FindScriptRun` function is used to identify blocks of subscript text in a string, taking into account script and language considerations, making use of tables in the string-manipulation (`'itl2'`) resource in its computations. Some script systems include subscripts, which are character sets that are subsidiary to the main character set. One useful subscript is the set of all character codes that have the same meaning in Roman as they do in a non-Roman script. For other scripts such as Japanese, there are additional useful subscripts. For example, a Japanese script system might include some Hiragana characters that are useful for input methods.

`FindScriptRun` computes the length of the current run of subscript text in the text string specified by `textPtr` and `textLen`. It assigns the length, in bytes, to the `lenUsed` parameter and returns a status code. You can advance the text pointer by the value of `lenUsed` to make subsequent calls to this function. You can use this function to identify runs of subscript characters so that you can treat them separately.

Word processors and other applications can call `FindScriptRun` to separate style runs of native text from non-native text. You can use this capability to extract those characters and apply a different font to them.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextUtils.h`

FindWordBreaks

Determines the beginning and ending boundaries of a word in a text string. (Deprecated in Mac OS X v10.4. Use `UCFindTextBreak` (page 2159) instead.)

```
void FindWordBreaks (
   Ptr textPtr,
   short textLength,
   short offset,
   Boolean leadingEdge,
   BreakTablePtr breaks,
   OffsetTable offsets,
   ScriptCode script
);
```

Parameters

textPtr

A pointer to the text string to be examined.

textLength

The number of bytes in the text string.

offset

A byte offset into the text. This parameter plus the `leadingEdge` parameter determine the position of the character at which to start the search.

leadingEdge

A flag that specifies which character should be used to start the search. If `leadingEdge` is `TRUE`, the search starts with the character specified in the `offset` parameter; if it is `FALSE`, the search starts with the character preceding the offset.

breaks

A pointer to a word-break table of type `NBreakTable` or `BreakTable`. If the value of this pointer is 0, the default word-break table of the script system specified by the `script` parameter is used. If the value of this pointer is –1, the default line-break table of the specified script system is used.

offsets

On output, the values in this table indicate the boundaries of the word that has been found.

script

The script code for the script system whose tables are used to determine where word boundaries occur.

Discussion

`FindWordBreaks` searches for a word in a text string, taking into account script and language considerations, making use of tables in the string-manipulation (`'itl2'`) resource in its computations. The `textPtr` and `textLength` parameters specify the text string that you want searched. The `offset` parameter and `leadingEdge` parameter together indicate where the search begins.

`FindWordBreaks` searches backward through the text string for one of the word boundaries and forward through the text string for its other boundary. It uses the definitions in the table specified by `nbreaks` to determine what constitutes the boundaries of a word. Each script system's word-break table is part of its string-manipulation (`'itl2'`) resource.

`FindWordBreaks` returns its results in an `OffsetTable` structure. `FindWordBreaks` uses only the first element of this three-element table. Each element is a pair of integers: `offFirst` and `offSecond`.

`FindWordBreaks` places the offset from the beginning of the text string to just before the leading edge of the character of the word that it finds in the `offFirst` field.

`FindWordBreaks` places the offset from the beginning of the text string to just after the trailing edge of the last character of the word that it finds in the `offSecond` field. For example, if the text "This is it" is passed with `offset` set to 0 and `leadingEdge` set to `TRUE`, then `FindWordBreaks` returns the offset pair (0,4).

If `leadingEdge` is `TRUE` and the value of `offset` is 0, then `FindWordBreaks` returns the offset pair (0,0). If `leadingEdge` is `FALSE` and the value of `offset` equals the value of `textLength`, then `FindWordBreaks` returns the offset pair with values (`textLength`, `textLength`).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextUtils.h`

FormatRecToString

Converts an internal representation of number formatting information into a number format specification string, which can be displayed and modified. (Deprecated in Mac OS X v10.4. Use `CFNumberFormatterGetFormat` instead.)

```
FormatStatus FormatRecToString (
   const NumFormatString *myCanonical,
   const NumberParts *partsTable,
   Str255 outString,
   TripleInt positions
);
```

Parameters

myCanonical
A pointer to the internal representation of number formatting information, as created by a previous call to the `StringToFormatRec` function.

partsTable
A pointer to a structure, obtained from the tokens (`'itl4'`) resource, that shows the correspondence between generic number part separators (tokens) and their localized version (for example, a thousand separator is a comma in the United States and a decimal point in France).

outString
On output, contains the number format specification string.

positions
An array that specifies the starting position and length of each of the three possible format strings (positive, negative, or zero) in the number format specification string. Semicolons are used as separators in the string.

Return Value
A value that denotes the confidence level for the conversion that it performed. The low byte of the `FormatStatus` value is of type `FormatResultType`. Be sure to cast the result of `FormatRecToString` to a type `FormatResultType` before working with it. See the description of the `FormatStatus` data type.

Discussion
`FormatRecToString` is the inverse operation of `StringToFormatRec` (page 2060). The internal representation of the formatting information in `myFormatRec` must have been created by a prior call to the `StringToFormatRec` function. The information in the number parts table specifies how to build the string representation.

The output number format specification string in `outString` specifies how numbers appear. This string contains three parts, which are separated by semicolons. The first part is the positive number format, the second is the negative number format, and the third part is the zero number format.

The `positions` parameter is an array of three integers (a `TripleInt` value), which specifies the starting position in `outString` of each of three formatting specifications:

- `positions[fPositive]`. The index in `outString` of the first byte of the formatting specification for positive number values.
- `positions[fNegative]`. The index in `outString` of the first byte of the formatting specification for negative number values.
- `positions[fZero]`. The index in `outString` of the first byte of the formatting specification for zero number values.

To obtain a handle to the number parts table from a tokens resource, use the `GetIntlResourceTable` 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
`NumberFormatting.h`

GetIndString

Loads a string from a string list (`'STR#'`) resource into memory, given the resource ID of the string list and the index of the individual string. (Deprecated in Mac OS X v10.4. Use `CFBundleCopyLocalizedString` instead.)

```
void GetIndString (
   Str255 theString,
   short strListID,
   short index
);
```

Parameters

`theString`
On output, the Pascal string result; specifically, a copy of the string from a string list that has the resource ID provided in the `strListID` parameter. If the resource that you specify cannot be read or the index that you specify is out of range for the string list, `GetIndString` sets `theString` to an empty string.

`strListID`
The resource ID of the `'STR#'` resource that contains the string list.

`index`
The index of the string in the list. This is a value from 1 to the number of strings in the list that is referenced by the `strListID` parameter.

Discussion
If necessary, `GetIndString` reads the string list from the resource file by calling the Resource Manager function `GetResource`. `GetIndString` accesses the string specified by the `index` parameter and copies it into `theString`.

Special Considerations

`GetIndString` 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
`TextUtils.h`

GetString

Loads a string from a string (`'STR'`) resource into memory. (Deprecated in Mac OS X v10.4. Use `CFBundleCopyLocalizedString` instead.)

```
StringHandle GetString (
   short stringID
);
```

Parameters

`stringID`
The resource ID of the string (`'STR '`) resource containing the string.

Return Value
A handle to a string with the specified resource ID. If necessary, `GetString` reads the handle form the resource file. If `GetString` cannot read the resource, it returns `NULL`.

Discussion
`GetString` calls the `GetResource` function of the Resource Manager to access the string. This means that if the specified resource is already in memory, `GetString` simply returns its handle.

Like the `NewString` (page 2051) function, `GetString` returns a handle whose size is based upon the actual length of the string.

If your application uses a large number of strings, it is more efficient to store them in a string list (`'STR#'`) resource than as individual resources in the resource file. You then use the `GetIndString` (page 2044) function to access each string in the list.

Special Considerations

`GetString` does not create a copy of the string.

`GetString` 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
`TextUtils.h`

IdenticalString

Compares two Pascal strings for equality, making use of the string comparison information from a resource that you specify as a parameter. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` instead.)

```
short IdenticalString (
   ConstStr255Param aStr,
   ConstStr255Param bStr,
   Handle itl2Handle
);
```

Parameters

aStr

One of the Pascal strings to be compared.

bStr

The other Pascal string to be compared.

itl2Handle

A handle to a string-manipulation (`'itl2'`) resource that contains string comparison information.

The `itl2Handle` parameter is used to specify a string-manipulation resource. If the value of this parameter is `NULL`, `IdenticalString` makes use of the resource for the current script. The string-manipulation resource includes tables for modifying string comparison and tables for case conversion and stripping of diacritical marks.

Return Value
Returns 0 if the two strings are equal; 1 if they are not equal. It compares the two strings without regard for secondary sorting order, the meaning of which depends on the language of the strings. For example, for the English language, using only primary differences means that `IdenticalString` ignores diacritical marks and does not distinguish between lowercase and uppercase. For example, if the two strings are `'Rose'` and `'rosé'`, `IdenticalString` considers them equal and returns 0.

Discussion
`IdenticalString` uses only primary differences in its comparison.

Special Considerations

`IdenticalString` 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
`StringCompare.h`

IdenticalText

Compares two text strings for equality, making use of the string comparison information from a resource that you specify as a parameter. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` instead.)

```
short IdenticalText (
   const void *aPtr,
   const void *bPtr,
   short aLen,
   short bLen,
   Handle itl2Handle
);
```

Parameters

aPtr

A pointer to the first character of the first text string.

bPtr

A pointer to the first character of the second text string.

aLen

The length, in bytes, of the first text string.

bLen

The length, in bytes, of the second text string.

itl2Handle

A handle to a string-manipulation (`'itl2'`) resource that contains string comparison information.

The `itl2Handle` parameter is used to specify a string-manipulation resource. If the value of this parameter is `NULL`, `IdenticalText` makes use of the resource for the current script. The string-manipulation resource includes functions and tables for modifying string comparison and tables for case conversion and stripping of diacritical marks.

Return Value

0 if the two text strings are equal; 1 if they are not equal. It compares the strings without regard for secondary sorting order, which means that it ignores diacritical marks and does not distinguish between lowercase and uppercase. For example, if the two text strings are `'Rose'` and `'rosé'`, `IdenticalText` considers them equal and returns 0.

Discussion

`IdenticalText` uses only primary sorting order in its comparison.

Special Considerations

`IdenticalText` 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

`StringCompare.h`

InvokeIndexToStringUPP

Call an index-to-string callback. (Deprecated in Mac OS X v10.4.)

```
Boolean InvokeIndexToStringUPP (
   short item,
   ScriptCode *itemsScript,
   StringPtr *itemsStringPtr,
   void *yourDataPtr,
   IndexToStringUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeIndexToStringUPP`, as the system calls your index-to-string callback function for you. See the callback `IndexToStringProcPtr` (page 2071) for more information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TypeSelect.h`

LanguageOrder

Determines the order in which strings in two different languages should be sorted. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` or `UCCompareText` (page 2151) instead.)

```
short LanguageOrder (
   LangCode language1,
   LangCode language2
);
```

Parameters

language1
The language code of the first language.

language2
The language code of the second language.

Return Value
A value that indicates the sorting order: –1 if strings in the first language should be sorted before sorting text in the second language, 1 if strings in the first language should be sorted after sorting strings in the second language, or 0 if the sorting order does not matter (that is, if the languages are the same).

Discussion
`LanguageOrder` takes a pair of language codes and determines in which order strings from the first language should be sorted relative to strings from the second language.

"Implicit Language Codes" (page 2082) are listed in the Constants section. The implicit language codes `scriptCurLang` and `scriptDefLang` are not valid for `LanguageOrder` because the script system being used is not specified as a parameter to this function.

Special Considerations

`LanguageOrder` 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
`StringCompare.h`

LowercaseText

Converts any uppercase characters in a text string into their lowercase equivalents. (Deprecated in Mac OS X v10.4. Use `CFStringLowercase` instead.)

```
void LowercaseText (
   Ptr textPtr,
   short len,
   ScriptCode script
);
```

Parameters

`textPtr`

A pointer to the text string to be converted.

`len`

The number of bytes in the text string. The text string can be up to 32 KB in length.

`script`

The script code for the script system whose resources are used to determine the results of converting characters.

The conversion uses tables in the string-manipulation (`'itl2'`) resource of the script specified by the value of the `script` parameter. You can specify `smSystemScript` to use the system script and `smCurrentScript` to use the script of the current font in the current graphics port.

Discussion
`LowercaseText` traverses the characters starting at the address specified by `textPtr` and continues for the number of characters specified in `len`. It converts any uppercase characters in the text into lowercase.

If `LowercaseText` cannot access the specified resource, it generates an error code and does not modify the string. You need to call the `ResError` function to determine which, if any, error occurred.

Special Considerations

`LowercaseText` 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
`TextUtils.h`

Munger

Searches text for a specified string pattern and replaces it with another string.

```
long Munger (
   Handle h,
   long offset,
   const void *ptr1,
   long len1,
   const void *ptr2,
   long len2
);
```

Parameters

h

A handle to the text string that is being manipulated.

offset

The byte offset in the destination string at which `Munger` begins its operation.

ptr1

A pointer to the first character in the string for which `Munger` is searching.

len1

The number of bytes in the string for which `Munger` is searching.

ptr2

A pointer to the first character in the substitution string.

len2

The number of bytes in the substitution string.

Return Value

A negative value if `Munger` cannot find the designated string.

Discussion

`Munger` manipulates bytes in a string to which you specify a handle in the `h` parameter. The manipulation begins at a byte offset, specified in `offset`, in the string. `Munger` searches for the string specified by `ptr1` and `len1`; when it finds an instance of that string, it replaces it with the substitution string, which is specified by `ptr2` and `len2`.

`Munger` operates on a byte-by-byte basis, which can produce inappropriate results for 2-byte script systems. The `ReplaceText` (page 2056) function works properly for all languages. You are encouraged to use `ReplaceText` instead of `Munger` whenever possible.

`Munger` takes special action if either of the specified pointer values is `NULL` or if either of the length values is `0`.

- If `ptr1` is `NULL`, `Munger` replaces characters without searching. It replaces `len1` characters starting at the `offset` location with the substitution string.
- If `ptr1` is `NULL` and `len1` is negative, `Munger` replaces all of the characters from the `offset` location to the end of the string with the substitution string.
- If `len1` is 0, `Munger` inserts the substitution string without replacing anything. `Munger` inserts the string at the `offset` location and returns the offset of the first byte past where the insertion occurred.
- If `ptr2` is `NULL`, `Munger` searches but does not replace. In this case, `Munger` returns the offset at which the string was found.
- If `len2` is 0 and `ptr2` is not `NULL`, `Munger` searches and deletes. In this case, `Munger` returns the offset at which it deleted.

- If the portion of the string from the `offset` location to its end matches the beginning of the string that `Munger` is searching for, `Munger` replaces that portion with the substitution string.

Be careful not to specify an offset with a value that is greater than the length of the destination string. Unpredictable results may occur.

`Munger` calls the `GetHandleSize` and `SetHandleSize` functions to access or modify the length of the string it is manipulating.

Special Considerations

`Munger` may move memory; your application should not call this function at interrupt time.

The destination string must be in a relocatable block that was allocated by the Memory Manager.

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

Declared In
`TextUtils.h`

NewIndexToStringUPP

Creates a new universal procedure pointer (UPP) to an index-to-string callback. (Deprecated in Mac OS X v10.4.)

```
IndexToStringUPP NewIndexToStringUPP (
   IndexToStringProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your index-to-string callback.

Return Value
On return, a UPP to the index-to-string callback.

Discussion
See the callback `IndexToStringProcPtr` (page 2071) for more information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TypeSelect.h`

NewString

Allocates memory in the heap for a string, copies its contents, and produces a handle for the heap version of the string. (Deprecated in Mac OS X v10.4. Use `CFStringCreateCopy` instead.)

```
StringHandle NewString (
   ConstStr255Param theString
);
```

Parameters

theString

A Pascal string that you want copied onto the heap.

Return Value

A handle to the newly allocated string. If the string cannot be allocated, `NewString` returns `NULL`. The size of the allocated string is based on the actual length of `theString`, which may not be 255 bytes.

Discussion

Before using Pascal string functions that can change the length of the string, it is a good idea to maximize the size of the string object on the heap. You can call either the `SetString` (page 2057) function or the Memory Manager function `SetHandleSize` to modify the string's size.

Special Considerations

`NewString` 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

`TextUtils.h`

NumToString

Converts a long integer value into a Pascal string. (Deprecated in Mac OS X v10.4. Use `CFStringCreateWithFormat` instead.)

```
void NumToString (
   long theNum,
   Str255 theString
);
```

Parameters

theNum

A long integer value. If the value of the number in the parameter `theNum` is negative, the string begins with a minus sign; otherwise, the sign is omitted.

theString

On output, contains the Pascal string representation of the number. Leading zeros are suppressed, except that a value of 0 produces the string "0". `NumToString` does not include thousand separators or decimal points in its formatted output.

Discussion

`NumToString` creates a string representation of `theNum` as a base-10 value and returns the result in `theString`.

Unless patched by a script system with different rules, this function assumes that you are using standard numeric token processing, meaning that the Roman script system number processing rules are used.

For functions that make use of the token-processing information that is found in the tokens (`'itl4'`) resource of script systems for converting numbers, see the sections "Using Number Format Specification Strings for International Number Formatting" and "Converting Between Strings and Floating-Point Numbers".

Special Considerations

`NumToString` 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
`NumberFormatting.h`

p2cstr

Converts a Pascal string to a C string. (Deprecated in Mac OS X v10.4. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

```
char * p2cstr (
   StringPtr aStr
);
```

Availability
Available in Mac OS X v10.4 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextUtils.h`

P2CStr

Converts a Pascal string to a C string. (Deprecated in Mac OS X v10.4. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

```
Ptr P2CStr (
    StringPtr pString
);
```

Availability
Available in Mac OS X v10.4 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextUtils.h`

p2cstrcpy

Converts a Pascal string to a C string. (Deprecated in Mac OS X v10.4. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

```
void p2cstrcpy (
   char *dst,
   ConstStr255Param src
);
```

Parameters

dst

On output, the C string.

src

The Pascal string you want to convert.

Discussion

This function allows in-place conversion. That is, the `src` and `dst` parameters can point to the same memory location.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextUtils.h`

RelString

Compares two Pascal strings using the string comparison rules of the Macintosh file system and returns a value that indicates the sorting order of the first string relative to the second string. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` or `UCCompareText` (page 2151) instead.)

```
short RelString (
   ConstStr255Param str1,
   ConstStr255Param str2,
   Boolean caseSensitive,
   Boolean diacSensitive
);
```

Parameters

str1

One of the Pascal strings to be compared.

str2

The other Pascal string to be compared.

caseSensitive

A flag that indicates how to handle case-sensitive information during the comparison. If the value of `caseSens` is `TRUE`, uppercase characters are distinguished from the corresponding lowercase characters. If it is `FALSE`, case information is ignored.

diacSensitive

A flag that indicates how to handle information about diacritical marks during the string comparison. If the value of `diacSensitive` is `TRUE`, characters with diacritical marks are distinguished from the corresponding characters without diacritical marks during the comparison. If it is `FALSE`, diacritical marks are ignored.

Return Value

Returns –1 if the first string is less than the second string, 0 if the two strings are equal, and 1 if the first string is greater than the second string. It compares the two strings in the same manner as does the `EqualString` function, by simply looking at the ASCII values of their characters. However, `RelString` provides more information about the two strings—it indicates their relationship to each other, rather than determining if they are exactly equal.

Discussion

This function does not make use of any script or language information; it assumes the original Macintosh character set only.

Special Considerations

The `RelString` function is not localizable and does not work properly with non-Roman 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.

Declared In

`StringCompare.h`

relstring

Compares two strings. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` or `UCCompareText` (page 2151) instead.)

Not recommended

```
short relstring (
   const char *str1,
   const char *str2,
   Boolean caseSensitive,
   Boolean diacSensitive
);
```

Parameters

str1

The string to be compared to `str2`.

str2

The string to be compared to `str1`.

caseSensitive

A flag that indicates how to handle case-sensitive information during the comparison.

diacSensitive

A flag that indicates how to handle information about diacritical marks during the string comparison.

Return Value
Returns –1 if the first string is less than the second string, 0 if the two strings are equal, and 1 if the first string is greater than the second string.

Discussion
This function is not recommended. Instead, see the function `RelString` (page 2054).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`StringCompare.h`

ReplaceText

Searches text on a character-by-character basis, replacing all instances of a string in that text with another string. (Deprecated in Mac OS X v10.4. Use `CFStringReplace` instead.)

```
short ReplaceText (
    Handle baseText,
    Handle substitutionText,
    Str15 key
);
```

Parameters

baseText
A handle to the string in which `ReplaceText` is to substitute text.

substitutionText
A handle to the string that `ReplaceText` uses as substitute text.

key
A Pascal string of less than 16 bytes that `ReplaceText` searches for.

Return Value
An integer value; if positive, it indicates the number of substitutions performed; if negative, it indicates an error. The constant `noErr` is returned if there was no error and no substitutions were performed.

Discussion
`ReplaceText` searches the text specified by the `baseText` parameter for instances of the string in the `key` parameter and replaces each instance with the text specified by the `substitutionText` parameter. `ReplaceText` searches on a character-by-character basis (as opposed to byte-by-byte), so it works properly for all script systems, including 2-byte script systems. It recognizes 2-byte characters in script systems that contain them and advances the search appropriately after encountering a 2-byte character.

Special Considerations

`ReplaceText` 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
StringCompare.h

ScriptOrder

Determines the order in which strings in two different scripts should be sorted. (Deprecated in Mac OS X v10.4. Use CFStringCompare or UCCompareText (page 2151) instead.)

```
short ScriptOrder (
   ScriptCode script1,
   ScriptCode script2
);
```

Parameters

script1
The script code of the first script.

script2
The script code of the second script.

Return Value
A value that indicates the sorting order: –1 if strings in the first script should be sorted before strings in the second script are sorted, 1 if strings in the first script should be sorted after strings in the second script are sorted, or 0 if the sorting order does not matter (that is, if the scripts are the same).

Discussion
Text of the system script is always first in a sorted list, regardless of the result returned by this function. When determining the order in which text from two different script systems should be sorted, the system script always sorts first, and scripts that are not enabled and installed always sort last. Invalid script or language codes always sort after valid ones.

Special Considerations

ScriptOrder 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
StringCompare.h

SetString

Changes the contents of a string referenced by a string handle, replacing the previous contents by copying the specified string. (Deprecated in Mac OS X v10.4. Use CFStringCreateWithPascalString and CFStringReplaceAll.)

```
void SetString (
   StringHandle theString,
   ConstStr255Param strNew
);
```

Parameters

theString

A Pascal string.

strNew

A handle to the string in memory whose contents you are replacing. If the new string (`theString`) is larger than the string originally referenced by `strNew`, `SetString` automatically resizes the handle and copies in the contents of the specified string.

Special Considerations

`SetString` 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

`TextUtils.h`

StringOrder

Compares two Pascal strings, taking into account the script system and language for each of the strings. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` or `UCCompareText` (page 2151) instead.)

```
short StringOrder (
   ConstStr255Param aStr,
   ConstStr255Param bStr,
   ScriptCode aScript,
   ScriptCode bScript,
   LangCode aLang,
   LangCode bLang
);
```

Parameters

aStr

One of the Pascal strings to be compared.

bStr

The other Pascal string to be compared.

aScript

The script code for the second string.

bScript

The script code for the first string.

aLang

The language code for the first string.

bLang

The language code for the second string.

Return Value
–1 if the first string is less than the second string, 0 if the first string is equal to the second string, and 1 if the first string is greater than the second string. The ordering of script and language codes, which is based on information in the script-sorting resource, is considered in determining the relationship of the two strings.

Discussion
This function takes both primary and secondary sorting orders into consideration and returns a value that indicates whether the first string is less than, equal to, or greater than the second string.

"Implicit Language Codes" (page 2082) are listed in the Constants section. Most applications specify the language code `scriptCurLang` for both the `aLang` and `bLang` values.

`StringOrder` first calls `ScriptOrder` (page 2057); if the result of `ScriptOrder` is not 0 (that is, if the strings use different scripts), `StringOrder` returns the same result.

`StringOrder` next calls `LanguageOrder` (page 2048); if the result of `LanguageOrder` is not 0 (that is, if the strings use different languages), `StringOrder` returns the same result.

At this point, `StringOrder` has two strings that are in the same script and language, so it compares them by using the sorting rules for that script and language, applying both the primary and secondary sorting orders. If that script is not installed and enabled, it uses the sorting rules specified by the system script or the font script, depending on the state of the international resources selection flag.

The `StringOrder` function is primarily used to insert Pascal strings in a sorted list; for sorting, rather than using this function, it may be faster to sort first by script and language by using the `ScriptOrder` and `LanguageOrder` functions, and then to call the `CompareString` (page 2035) function, to sort strings within a script or language group.

Special Considerations

`StringOrder` 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
`StringCompare.h`

StringToExtended

Converts a string representation of a number into a floating-point number, using a `NumFormatStringRec` structure to specify how the input number string is formatted. (Deprecated in Mac OS X v10.4. Use `CFNumberFormatterCreateStringWithNumber` instead.)

```
FormatStatus StringToExtended (
   ConstStr255Param source,
   const NumFormatString *myCanonical,
   const NumberParts *partsTable,
   extended80 *x
);
```

Parameters

source

A Pascal string that contains the string representation of a number.

myCanonical

A pointer to the internal representation of the formatting information for numbers, as produced by the `StringToFormatRec` function.

partsTable

A pointer to a structure, obtained from the tokens (`'itl4'`) resource, that shows the correspondence between generic number part separators (tokens) and their localized version (for example, a thousand separator is a comma in the United States and a decimal point in France).

x

On output, contains a pointer to the 80-bit SANE representation of the floating-point number.

Return Value

A value that denotes the confidence level for the conversion that it performed. The low byte of the `FormatStatus` value is of type `FormatResultType`. Be sure to cast the result of `StringToExtended` to a type `FormatResultType` before working with it. `StringToExtended` returns an 80-bit, not a 96-bit, representation. See the description of the `FormatStatus` data type.

Discussion

`StringToExtended` uses the internal representation of number formatting information that was created by a prior call to `StringToFormatRec` to parse the input number string. It uses the number parts table to determine the components of the number string that is being converted. `StringToExtended` parses the string and then converts the string to a simple form, stripping nondigits and replacing the decimal point before converting it into a floating-point number. If the input string does not match any of the patterns, then `StringToExtended` parses the string as well as it can and returns a confidence level result that indicates the parsing difficulties.

To obtain a handle to the number parts table from a tokens resource, use the `GetIntlResourceTable` 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

`NumberFormatting.h`

StringToFormatRec

Creates a number format specification string structure from a number format specification string that you supply in a Pascal string. (Deprecated in Mac OS X v10.4. Use `CFNumberFormatterSetFormat` instead.)

```
FormatStatus StringToFormatRec (
   ConstStr255Param inString,
   const NumberParts *partsTable,
   NumFormatString *outString
);
```

Parameters

inString

A Pascal string that contains the number formatting specification.

The `inString` parameter contains a number format specification string that specifies how numbers appear. This string contains up to three specifications, separated by semicolons. The positive number format is specified first, the negative number format is second, and the zero number format is last. If the string contains only one part, that is the format of all three types of numbers. If the string contains two parts, the first part is the format for positive and zero number values, and the second part is the format for negative numbers.

partsTable

A pointer to a structure, usually obtained from the tokens (`'itl4'`) resource, that shows the correspondence between generic number part separators (tokens) and their localized version (for example, a thousand separator is a comma in the United States and a decimal point in France).

outString

On output, a pointer to a `NumFormatStringRec` structure that contains the values that form the internal representation of the format specification. The format of the data in this structure is private.

Return Value

A value that denotes the confidence level for the conversion that was performed. The low byte of the value is of type `FormatResultType`. Be sure to cast the result of `StringToFormatRec` to a type `FormatResultType` before working with it. See the description of the `FormatStatus` data type.

Discussion

`StringToFormatRec` converts a number format specification string into the internal representation contained in a number format string structure. It uses information in the current script's tokens resource to determine the components of the number. `StringToFormatRec` checks the validity both of the input format string and of the number parts table (since this table can be programmed by the application). `StringToFormatRec` ignores spurious characters.

To obtain a handle to the number parts table from a tokens resource, use the `GetIntlResourceTable` function.

Special Considerations

`StringToFormatRec` 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

`NumberFormatting.h`

StringToNum

Converts the Pascal string representation of a base-10 number into a long integer value. (Deprecated in Mac OS X v10.4. Use `CFStringGetIntValue` instead.)

```
void StringToNum (
   ConstStr255Param theString,
   long *theNum
);
```

Parameters

theString

A Pascal string representation of a base-10 number. The numeric string can be padded with leading zeros or with a sign.

theNum

On output, contains a pointer to the numeric value.

Discussion

Unless patched by a script system with different rules, this function assumes that you are using standard numeric token processing, meaning that the Roman script system number processing rules are used.

For functions that make use of the token-processing information that is found in the tokens (`'itl4'`) resource of script systems for converting numbers, see the sections "Using Number Format Specification Strings for International Number Formatting" and "Converting Between Strings and Floating-Point Numbers".

The 32-bit result is negated if the string begins with a minus sign. Integer overflow occurs if the magnitude is greater than or equal to 2 raised to the 31st power. `StringToNum` performs the negation using the two's complement method: the state of each bit is reversed and then 1 is added to the result. For example, here are possible results produced by `StringToNum`:

- The value of `theString` is "-23". `StringToNum` returns the value -23 in `theNum`.
- The value of `theString` is "-0". `StringToNum` returns the value 0 in `theNum`.
- The value of `theString` is "055". `StringToNum` returns the value 55 in `theNum`.
- The value of `theString` is "2147483648" (magnitude is 2^31). `StringToNum` returns the value –2147483648 in `theNum`.
- The value of `theString` is "–2147483648". `StringToNum` returns the value –2147483648 in `theNum`.
- The value of `theString` is "4294967295" (magnitude is 2^32–1). `StringToNum` returns the value -1 in `theNum`.
- The value of `theString` is "–4294967295". `StringToNum` returns the value 1 in `theNum`.

`StringToNum` does not check whether the characters in the string are between 0 and 9; instead, it takes advantage of the fact that the ASCII values for these characters are $30 through $39, and masks the last four bits for use as a digit. For example, `StringToNum` converts 2: to the number 30 since the character code for the colon (:) is $3A. Because `StringToNum` operates this way, spaces are treated as zeros (the character code for a space is $20), and other characters do get converted into numbers. For example, the character codes for 'C', 'A', and 'T' are $43, $41, and $54 respectively. Hence, the strings 'CAT', '+CAT', and '-CAT' would produce the results 314, 314, and -314.

One consequence of this conversion method is that `StringToNum` does not ignore thousand separators (the "," character in the United States), which can lead to improper conversions. It is a good idea to ensure that all characters in `theString` are valid digits before you call `StringToNum`.

Special Considerations

`StringToNum` 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

`NumberFormatting.h`

StripDiacritics

Strips any diacritical marks from a text string. (Deprecated in Mac OS X v10.4. Use `CFStringTransform` instead.)

```
void StripDiacritics (
   Ptr textPtr,
   short len,
   ScriptCode script
);
```

Parameters

`textPtr`

A pointer to the text string to be stripped.

`len`

The length, in bytes, of the text string. The text string can be up to 32 KB in length.

`script`

The script code for the script system whose rules are used for determining which character results from stripping a diacritical mark.

The conversion uses tables in the string-manipulation (`'itl2'`) resource of the script specified by the value of the `script` parameter. You can specify `smSystemScript` to use the system script and `smCurrentScript` to use the script of the current font in the current graphics port.

Discussion

`StripDiacritics` traverses the characters starting at the address specified by `textPtr` and continues for the number of characters specified in `len`. It strips any diacritical marks from the text.

If `StripDiacritics` cannot access the specified resource, it generates an error code and does not modify the string. You need to call the `ResError` function to determine which, if any, error occurred.

Special Considerations

`StripDiacritics` 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

`TextUtils.h`

TextOrder

Compares two text strings, taking into account the script and language for each of the strings. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` or `UCCompareText` (page 2151) instead.)

```
short TextOrder (
   const void *aPtr,
   const void *bPtr,
   short aLen,
   short bLen,
   ScriptCode aScript,
   ScriptCode bScript,
   LangCode aLang,
   LangCode bLang
);
```

Parameters

aPtr
A pointer to the first character of the first text string.

bPtr
A pointer to the first character of the second text string.

aLen
The length, in bytes, of the first text string.

bLen
The length, in bytes, of the second text string.

aScript
The script code for the first text string.

bScript
The script code for the second text string.

aLang
The language code for the first text string.

bLang
The language code for the second text string.

Return Value

Returns –1 if the first string is less than the second string, 0 if the first string is equal to the second string, and 1 if the first string is greater than the second string. The ordering of script and language codes, which is based on information in the script-sorting resource, is considered in determining the relationship of the two strings.

Discussion

This function takes both primary and secondary sorting orders into consideration and returns a value that indicates whether the first string is less than, equal to, or greater than the second string.

"Implicit Language Codes" (page 2082) are listed in the Constants section. Most applications specify the language code `scriptCurLang` for both the `aLang` and `bLang` values.

`TextOrder` first calls `ScriptOrder` (page 2057); if the result of `ScriptOrder` is not 0 (that is, if the strings use different scripts), `TextOrder` returns the same result.

`TextOrder` next calls `LanguageOrder` (page 2048); if the result of `LanguageOrder` is not 0 (that is, if the strings use different languages), `TextOrder` returns the same result.

At this point, `TextOrder` has two strings that are in the same script and language, so it compares them by using the sorting rules for that script and language, applying both the primary and secondary sorting orders. If that script is not installed and enabled, it uses the sorting rules specified by the system script or the font script, depending on the state of the international resources selection flag.

The `TextOrder` function is primarily used to insert text strings in a sorted list; for sorting, rather than using this function, it may be faster to sort first by script and language by using the `ScriptOrder` and `LanguageOrder` functions, and then to call the `CompareText` (page 2036) function, to sort strings within a script or language group.

Special Considerations

`TextOrder` 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
`StringCompare.h`

TypeSelectClear

Clears the key list and resets the type select record. (Deprecated in Mac OS X v10.4. Use `UCTypeSelectFlushSelectorData` instead.)

Not recommended.

```
void TypeSelectClear (
   TypeSelectRecord *tsr
);
```

Parameters

tsr

A pointer to the type-select record you want to clear.

Discussion
The use of this function is not recommended in a Unicode-based application. If you want to use this function in an application that uses the Unicode character set, you must first convert Unicode text strings to Macintosh encoded Pascal text strings. You must also provide the encoding type or be able to determine it by extracting it from the text or by examining the system or keyboard script.

Special Considerations

For Unicode-based applications, you should use the UCTypeSelect functions, which manipulate a `UCTypeSelectRef` object. For more detalis, see Unicode Utilities (`UnicodeUtilities.h`).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TypeSelect.h`

TypeSelectCompare

Compares a text buffer to the keystroke buffer. (Deprecated in Mac OS X v10.4. Use `UCTypeSelectCompare` instead.)

Not recommended.

```
short TypeSelectCompare (
   const TypeSelectRecord *tsr,
   ScriptCode testStringScript,
   StringPtr testStringPtr
);
```

Parameters

tsr

A type select record that contains the keystroke buffer.

testStringScript

The script code of the test string.

testStringPtr

A pointer to the text you want to compare to the keystroke buffer.

Return Value

A numerical value that represents the ordering of the characters in the keystroke buffer with respect to the test string buffer. The value -1 is returned if characters in the keystroke buffer sort before those in `testStringPtr`; 0 if characters in the keystroke buffer are the same as those in `testStringPtr`, and 1 if the characters in the keystroke buffer sort after those in `testStringPtr`.

Discussion

The use of this function is not recommended in a Unicode-based application. If you want to use this function in an application that uses the Unicode character set, you must first convert Unicode text strings to Macintosh encoded Pascal text strings. You must also provide the encoding type or be able to determine it by extracting it from the text or by examining the system or keyboard script.

Special Considerations

For Unicode-based applications, you should use the UCTypeSelect functions, which manipulate a `UCTypeSelectRef` object. For more detalis, see Unicode Utilities (`UnicodeUtilities.h`).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TypeSelect.h`

TypeSelectFindItem

Finds the closest match between a specified list of characters and the keystrokes stored in the type select record. (Deprecated in Mac OS X v10.4. Use `UCTypeSelectFindItem` instead.)

Not recommended.

```
short TypeSelectFindItem (
   const TypeSelectRecord *tsr,
   short listSize,
   TSCode selectMode,
   IndexToStringUPP getStringProc,
   void *yourDataPtr
);
```

Parameters

tsr

A pointer to the type select record that contains the keystrokes you want to compare.

listSize

The size of the list to search through.

selectMode

The select mode. See `Type Select Modes` (page 2083)for a list of the constants you can supply.

getStringProc

A pointer to your index-to-string callback function. See `IndexToStringProcPtr` (page 2071) for more information.

yourDataPtr

A pointer to your data structure. This is passed to your index-to-string callback, and can be `NULL`, depending on how you implement your callback function.

Return Value

Discussion

The use of this function is not recommended in a Unicode-based application. If you want to use this function in an application that uses the Unicode character set, you must first convert Unicode text strings to Macintosh encoded Pascal text strings. You must also provide the encoding type or be able to determine it by extracting it from the text or by examining the system or keyboard script.

Special Considerations

For Unicode-based applications, you should use the UCTypeSelect functions, which manipulate a `UCTypeSelectRef` object. For more detalis, see Unicode Utilities (`UnicodeUtilities.h`).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TypeSelect.h`

TypeSelectNewKey

Creates a new type select record. (Deprecated in Mac OS X v10.4. Use `UCTypeSelectCreateSelector` instead.)

Not recommended.

```
Boolean TypeSelectNewKey (
   const EventRecord *theEvent,
   TypeSelectRecord *tsr
);
```

Parameters

theEvent

A pointer to an event record.

tsr

A pointer to a type select record.

Return Value

Returns `true` if the function executed successfully; `false` otherwise.

Discussion

The use of this function is not recommended in a Unicode-based application. If you want to use this function in an application that uses the Unicode character set, you must first convert Unicode text strings to Macintosh encoded Pascal text strings. You must also provide the encoding type or be able to determine it by extracting it from the text or by examining the system or keyboard script.

Special Considerations

For Unicode-based applications, you should use the UCTypeSelect functions, which manipulate a `UCTypeSelectRef` object. For more detalis, see Unicode Utilities (`UnicodeUtilities.h`).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TypeSelect.h`

UppercaseStripDiacritics

Converts any lowercase characters in a text string into their uppercase equivalents and strips any diacritical marks from the text. (Deprecated in Mac OS X v10.4. Use `CFStringTransform` instead.)

```
void UppercaseStripDiacritics (
   Ptr textPtr,
   short len,
   ScriptCode script
);
```

Parameters

textPtr

A pointer to the text string to be converted.

len

The length, in bytes, of the text string. The text string can be up to 32 KB in length.

script

The script code of the script system whose resources are used to determine the results of converting characters.

The conversion uses tables in the string-manipulation (`'itl2'`) resource of the script specified by the value of the `script` parameter. You can specify `smSystemScript` to use the system script and `smCurrentScript` to use the script of the current font in the current graphics port.

Discussion

`UppercaseStripDiacritics` traverses the characters starting at the address specified by `textPtr` and continues for the number of characters specified in `len`. It converts lowercase characters in the text into their uppercase equivalents and also strips diacritical marks from the text string. This function combines the effects of the `UppercaseText` (page 2069) and `StripDiacritics` (page 2063) functions.

If `UppercaseStripDiacritics` cannot access the specified resource, it generates an error code and does not modify the string. You need to call the `ResError` function to determine which, if any, error occurred.

Special Considerations

`UppercaseStripDiacritics` 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

`TextUtils.h`

UppercaseText

Converts any lowercase characters in a text string into their uppercase equivalents. (Deprecated in Mac OS X v10.4. Use `CFStringUppercase` instead.)

```
void UppercaseText (
   Ptr textPtr,
   short len,
   ScriptCode script
);
```

Parameters

textPtr

A pointer to the text string to be converted.

len

The length, in bytes, of the text string. The text string can be up to 32 KB in length.

script

The script code of the script system whose case conversion rules are used for determining uppercase character equivalents.

The conversion uses tables in the string-manipulation (`'itl2'`) resource of the specified `script`. You can specify `smSystemScript` to use the system script and `smCurrentScript` to use the script of the current font in the current graphics port.

Discussion

`UppercaseText` traverses the characters starting at the address specified by `textPtr` and continues for the number of characters specified in `len`. It converts any lowercase characters in the text into uppercase.

If `UppercaseText` cannot access the specified resource, it generates an error code and does not modify the string. You need to call the `ResError` function to determine which, if any, error occurred.

Special Considerations

`UppercaseText` 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

`TextUtils.h`

UpperString

Converts any lowercase letters in a Pascal string to their uppercase equivalents, using the Macintosh file system rules. (Deprecated in Mac OS X v10.4. Use `CFStringUppercase` instead.)

```
void UpperString (
   Str255 theString,
   Boolean diacSensitive
);
```

Parameters

theString

On input, this is the Pascal string to be converted. On output, this contains the string resulting from the conversion. `UpperString` traverses the characters in `theString` and converts any lowercase characters with character codes in the range 0x00 through 0xD8 into their uppercase equivalents. `UpperString` places the converted characters in `theString`.

diacSensitive

A flag that indicates whether the case conversion is to strip diacritical marks. If the value of this parameter is `TRUE`, diacritical marks are considered in the conversion; if it is `FALSE`, any diacritical marks are stripped.

Discussion

Only a subset of the Roman character set (character codes with values through $D8) are converted. Use this function to emulate the behavior of the Macintosh file 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
`TextUtils.h`

upperstring

Converts any lowercase letters in a Pascal string to their uppercase equivalents. (Deprecated in Mac OS X v10.4. Use `CFStringUppercase` instead.)

Not recommended

```
void upperstring (
   char *theString,
   Boolean diacSensitive
);
```

Parameters

`theString`

On input, this is the Pascal string to be converted. On output, this contains the string resulting from the conversion.

`diacSensitive`

A flag that indicates whether the case conversion is to strip diacritical marks. If the value of this parameter is `TRUE`, diacritical marks are considered in the conversion; if it is `FALSE`, any diacritical marks are stripped.

Discussion
You should use the function `CFStringUppercase` instead of this one.

Carbon Porting Notes

Use `UpperString` instead.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextUtils.h`

Callbacks

IndexToStringProcPtr

Defines a pointer to your index-to-string callback function that retrieves the string associated with an index value.

Not recommended.

```
typedef Boolean (*IndexToStringProcPtr)
(
    short item,
    ScriptCode * itemsScript,
    StringPtr * itemsStringPtr,
    void * yourDataPtr
);
```

If you name your function `MyIndexToStringProc`, you would declare it like this:

```
Boolean MyIndexToStringProcPtr (
    short item,
    ScriptCode * itemsScript,
    StringPtr * itemsStringPtr,
    void * yourDataPtr
);
```

Parameters

item
The index value for which the `TypeSelect` function requests a string.

itemsScript
The script code of the string specified by `itemsStringPtr`.

itemsStringPtr
On return, points to the string that matches the index specify by the `item` parameter.

yourDataPtr
A pointer to your data structure. This is passed to your index-to-string callback, and can be `NULL`, depending on how you implement your callback function.

Return Value
Returns `true` if a string matching that index value was found; `false` otherwise.

Discussion
The use of this function is not recommended in a Unicode-based application. If you want to use this function in an application that uses the Unicode character set, you must first convert Unicode text strings to Macintosh encoded Pascal text strings. You must also provide the encoding type or be able to determine it by extracting it from the text or by examining the system or keyboard script.

Availability
Not recommended. Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`TypeSelect.h`

Data Types

BreakTable

Contains information used to determine the boundaries of a word.

```
struct BreakTable {
    char charTypes[256];
    short tripleLength;
    short triples[1];
};
typedef struct BreakTable BreakTable;
typedef BreakTable * BreakTablePtr;
```

Discussion
You can supply a `BreakTable` as a parameter to the function `FindWordBreaks` (page 2041).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`TextUtils.h`

FormatClass

Defines a data type used to access entries in a triple integer array.

```
typedef SInt8 FormatClass;
```

Discussion
Each of the three `FVector` entries in a triple integer array is accessed by one of the values of the `FormatClass` type. See `FVector` (page 2073) for more information.

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

Declared In
`NumberFormatting.h`

FormatStatus

Defines a data type used to denote the confidence level for a conversion.

```
typedef short FormatStatus;
```

Discussion
A `FormatStatus` value is returned by the functions `ExtendedToString` (page 2039), `StringToExtended` (page 2059), `FormatRecToString` (page 2043), and `StringToFormatRec` (page 2060).

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

Declared In
`NumberFormatting.h`

FVector

Contains position and length information for one portion of a formatted numeric string.

```
struct FVector {
    short start;
    short length;
};
typedef struct FVector FVector;
typedef FVector TripleInt[3];
```

Fields

`start`

The starting byte position in the string of the specification information.

`length`

The number of bytes used in the string for the specification information.

Discussion

The `FVector` data structure is used in the `TripleInt` (page 2078) array.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NumberFormatting.h`

IndexToStringUPP

Defines a universal procedure pointer to an index-to-string callback.

```
typedef IndexToStringProcPtr IndexToStringUPP;
```

Discussion

For more information, see the description of the `IndexToStringProcPtr` (page 2071) callback function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`TypeSelect.h`

NBreakTable

Contains information used by the `FindWordBreaks` function to determine word boundaries.

```
struct NBreakTable {
    SInt8 flags1;
    SInt8 flags2;
    short version;
    short classTableOff;
    short auxCTableOff;
    short backwdTableOff;
    short forwdTableOff;
    short doBackup;
    short length;
    char charTypes[256];
    short tables[1];
};
typedef struct NBreakTable NBreakTable;
typedef NBreakTable * NBreakTablePtr;
```

Fields

`flags1`
: The high-order byte of the break table format flags. If the high-order bit of this byte is set to 1, this break table is in the format used by `FindWordBreaks`.

`flags2`
: The low-order byte of the break table format flags. If the value in this byte is 0, the break table belongs to a 1-byte script system; in this case `FindWordBreaks` does not check for 2-byte characters.

`version`
: The version of this break table.

`classTableOff`
: The offset in bytes from the beginning of the break table to the beginning of the class table.

`auxCTableOff`
: The offset in bytes from the beginning of the break table to the beginning of the auxiliary class table.

`backwdTableOff`
: The offset in bytes from the beginning of the break table to the beginning of the backward-processing table.

`forwdTableOff`
: The offset in bytes from the beginning of the break table to the beginning of the forward-processing table.

`doBackup`
: The minimum byte offset into the buffer for doing backward processing. If the selected character for `FindWordBreaks` has a byte offset less than `doBackup`, `FindWordBreaks` skips backward processing altogether and starts from the beginning of the buffer.

`length`
: The length in bytes of the entire break table, including the individuals tables.

`charTypes`
: The class table.

`tables`
: The data of the auxiliary class table, backward table, and forward table.

Discussion

The tables have this format and content:

- The class table is an array of 256 signed bytes. Offsets into the table represent byte values; if the entry at a given offset in the table is positive, it means that a byte whose value equals that offset is a single-byte character, and the entry at that offset is the class number for the character. If the entry is negative, it means that the byte is the first byte of a 2-byte character code, and the auxiliary class table must be used to determine the character class. Odd negative numbers are handled differently from even negative numbers.
- The auxiliary class table assigns character classes to 2-byte characters. It is used when the class table determines that a byte value represents the first byte of a 2-byte character.
  - The auxiliary class table handles odd negative values from the class table as follows. If the first word of the auxiliary class table is equal to or greater than zero, it represents the default class number for 2-byte character codes—the class assigned to every odd negative value from the class table. If the first word is less than zero, it is the negative of the offset from the beginning of the auxiliary class table to a first-byte class table (a table of 2-byte character classes that can be determined from just the first byte). The value from the class table is negated, 1 is subtracted from it to obtain an even offset, and the value at that offset into the first-byte class table is the class to be assigned.
  - The auxiliary class table handles even negative values from the class table as follows. The auxiliary class table begins with a variable-length word array. The words that follow the first word are offsets to row tables. Row tables have the same format as the class table, but are used to map the second byte of a 2-byte character code to a class number. If the entry in the class table for a given byte is an even negative number, `FindWordBreaks` negates this value to obtain the offset from the beginning of the auxiliary class table to the appropriate word, which in turn contains an offset to the appropriate row table. That row table is then used to map the second byte of the character to a class number.
- The backward-processing table is a state table used by `FindWordBreaks` for backward searching. Using the backward-processing table, `FindWordBreaks` starts at a specified character, moving backward as necessary until it encounters a word boundary.
- The forward-processing table is a state table used by `FindWordBreaks` for forward searching. Using the forward-processing table, `FindWordBreaks` starts at one word boundary and moves forward until it encounters another word boundary.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`TextUtils.h`

NumFormatString

Contains data that represents the internal number formatting specification.

```
struct NumFormatString {
    UInt8 fLength;
    UInt8 fVersion;
    char data[254];
};
typedef struct NumFormatString NumFormatString;
typedef NumFormatString NumFormatStringRec;
```

Fields

`fLength`
The number of bytes in the data actually used for this number formatting specification.

`fVersion`
The version number of the number formatting specification.

`data`
The data that comprises the number formatting specification.

Discussion

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

Declared In
`NumberFormatting.h`

NumFormatStringRec

Defines an internal numeric representation that is independent of region, language, and other multicultural consideration.

```
typedef NumFormatString NumFormatStringRec;
```

Discussion
To allow for all of the international variations in numeric presentation styles, you need to include in your function calls a number parts table from a tokens (`'itl4'`) resource. You can usually use the number parts table in the standard tokens resource that is supplied with the system. You also need to define the format of input and output numeric strings, including which characters (if any) to use as thousand separators, whether to indicate negative values with a minus sign or by enclosing the number in parentheses, and how to display zero values.

To make it possible to map a number that was formatted for one specification into another format, the Mac OS defines an internal numeric representation that is independent of region, language, and other multicultural considerations: the `NumFormatStringRec` structure. This structure is created from a number format specification string that defines the appearance of numeric strings.

Four of the numeric string functions use the number formatting specification, defined by the `NumFormatStringRec` data type: `StringToFormatRec` (page 2060), `FormatRecToString` (page 2043), `StringToExtended` (page 2059), and `ExtendedToString` (page 2039). The number format specification structure contains the data that represents the internal number formatting specification information. This data is stored in a private format.

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

Declared In
`NumberFormatting.h`

ScriptRunStatus

Contains script-specific information for a script run.

```
struct ScriptRunStatus {
    SInt8 script;
    SInt8 runVariant;
};
typedef struct ScriptRunStatus ScriptRunStatus;
```

Fields

`script`

The script code of the subscript run. Zero indicates the Roman script system.

`runVariant`

Script-specific information about the run, in the same format as that returned by the `CharacterType` function. This information includes the type of subscript—for example, Kanji, Katakana, or Hiragana for a Japanese script system.

Discussion

The `FindScriptRun` (page 2040) function returns the script run status structure, defined by the `ScriptRunStatus` data type, when it completes its processing, which is to find a run of subscript text in a string.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`TextUtils.h`

TripleInt

Defines a data type used to return the position and length information for three different portions of a formatted numeric string.

```
typedef FVector TripleInt[3];
```

Discussion

The `FormatRecToString` (page 2043) function uses the triple-integer array, defined by the `TripleInt` data type, to return the starting position and length in a string of three different portions of a formatted numeric string: the positive value string, the negative value string, and the zero value string. Each element of the triple integer array is an `FVector` structure. Each of the three `FVector` entries in the triple integer array is accessed by one of the values of the `FormatClass` type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NumberFormatting.h`

TypeSelectRecord

Contains a buffer of keystrokes, the script code associated with the keystrokes, and timer information.

```
struct TypeSelectRecord {
    unsigned long tsrLastKeyTime;
    ScriptCode tsrScript;
    Str63 tsrKeyStrokes;
};
typedef struct TypeSelectRecord TypeSelectRecord;
```

Fields

`tsrLastKeyTime`
A value that indicates timeout information.

`tsrScript`
A script code.

`tsrKeyStrokes`
The keystroke buffer.

Discussion

The `TypeSelectRecord` data structure is passed as a parameter to the functions `TypeSelectNewKey` (page 2067), `TypeSelectFindItem` (page 2066), `TypeSelectCompare` (page 2066), and `TypeSelectClear` (page 2065).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TypeSelect.h`

Constants

Format Result Types

Specify values that can be returned in the low byte of a format status (`FormatStatus`) value.

```
enum {
    fFormatOK = 0,
    fBestGuess = 1,
    fOutOfSynch = 2,
    fSpuriousChars = 3,
    fMissingDelimiter = 4,
    fExtraDecimal = 5,
    fMissingLiteral = 6,
    fExtraExp = 7,
    fFormatOverflow = 8,
    fFormStrIsNAN = 9,
    fBadPartsTable = 10,
    fExtraPercent = 11,
    fExtraSeparator = 12,
    fEmptyFormatString = 13
};
typedef SInt8 FormatResultType;
```

Constants

`fFormatOK`
Specifies format is okay.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fBestGuess`
Specifies the format is the best guess.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fOutOfSynch`
Specifies the format is out of sync.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fSpuriousChars`
Specifies the format contains spurious characters.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fMissingDelimiter`
Specifies a missing delimiter.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fExtraDecimal`
Specifies the format contains an extra decimal sign.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fMissingLiteral`
Specifies the format is missing a literal.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fExtraExp`
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fFormatOverflow`
Specifies a format overflow.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fFormStrIsNAN`
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fBadPartsTable`
Specifies the parts table is bad.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fExtraPercent`
Specifies the format contains an extra percent sign.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fExtraSeparator`
Specifies an extra separator.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fEmptyFormatString`
Specifies the format string is empty.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

Discussion
A format result type is returned in the low byte of a format status (`FormatStatus`) value. A `FormatStatus` (page 2073) value is returned by the functions `ExtendedToString` (page 2039), `StringToExtended` (page 2059), `FormatRecToString` (page 2043), and `StringToFormatRec` (page 2060). A format status value denotes the confidence level for a conversion.

TripleInt Index Values

Specify an index for a `TripleInt` array.

```
enum {
    fPositive = 0,
    fNegative = 1,
    fZero = 2
};
```

Constants

`fPositive`

Specifies the positive value string.

Available in Mac OS X v10.0 and later.

Declared in `NumberFormatting.h`.

`fNegative`

Specifies the negative value string.

Available in Mac OS X v10.0 and later.

Declared in `NumberFormatting.h`.

`fZero`

Specifies the zero value string.

Available in Mac OS X v10.0 and later.

Declared in `NumberFormatting.h`.

Discussion

See `TripleInt` (page 2078) for more information.

NumFormatString Version

Specifies the first version of the `NumFormatString` data structure.

```
enum {
    fVNumber = 0
};
```

Discussion

See `NumFormatString` (page 2076) for more information.

Implicit Language Codes

Specify implicit language codes.

```
enum {
    systemCurLang = -2,
    systemDefLang = -3,
    currentCurLang = -4,
    currentDefLang = -5,
    scriptCurLang = -6,
    scriptDefLang = -7
};
```

Constants

`systemCurLang`
Specifies the current language for system script (from `'itlb'`).
Available in Mac OS X v10.0 and later.
Declared in `StringCompare.h`.

`systemDefLang`
Specifies the default language for system script (from `'itlm'`).
Available in Mac OS X v10.0 and later.
Declared in `StringCompare.h`.

`currentCurLang`
Specifies the current language for current script (from `'itlb'`).
Available in Mac OS X v10.0 and later.
Declared in `StringCompare.h`.

`currentDefLang`
Specifies the default language for current script (from `'itlm'`).
Available in Mac OS X v10.0 and later.
Declared in `StringCompare.h`.

`scriptCurLang`
Specifies the current language for specified script (from `'itlb'`)
Available in Mac OS X v10.0 and later.
Declared in `StringCompare.h`.

`scriptDefLang`
Specifies the default language for specified script (from `'itlm'`)
Available in Mac OS X v10.0 and later.
Declared in `StringCompare.h`.

Discussion
The functions `LanguageOrder` (page 2048), `StringOrder` (page 2058), and `TextOrder` (page 2064) accept as parameters implicit language codes listed here, as well as explicit language codes.

Type Select Modes

Contains type-select code information.

```
typedef SInt16 TSCode;
enum {
    tsPreviousSelectMode = -1,
    tsNormalSelectMode = 0,
    tsNextSelectMode = 1
};
```

Constants

`tsPreviousSelectMode`
Specifies previous-select mode.
Available in Mac OS X v10.0 and later.
Declared in `TypeSelect.h`.

`tsNormalSelectMode`
Specifies normal-select mode.
Available in Mac OS X v10.0 and later.
Declared in `TypeSelect.h`.

`tsNextSelectMode`
Specifies next-select mode.
Available in Mac OS X v10.0 and later.
Declared in `TypeSelect.h`.

Discussion
This structure is passed as a parameter to the function `TypeSelectFindItem` (page 2066).

Obsolete Language Code Values

Specify language code values that are no longer used.

```
enum {
    iuSystemCurLang = systemCurLang,
    iuSystemDefLang = systemDefLang,
    iuCurrentCurLang = currentCurLang,
    iuCurrentDefLang = currentDefLang,
    iuScriptCurLang = scriptCurLang,
    iuScriptDefLang = scriptDefLang
};
```

Thread Manager Reference

Framework: CoreServices/CoreServices.h

Declared in Threads.h

Overview

You can use the Thread Manager to provide cooperatively scheduled threads, or multiple points of execution, in an application. You can think of the Thread Manager as an enhancement to the classic Mac OS Process Manager, which governs how applications work together in the Mac OS cooperative multitasking environment.

Important: Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Consider using the Thread Manager for applications with more than one thread if these threads can execute only in the cooperative multitasking environment of the classic Mac OS Process Manager.

Alternatively, you should consider using the Multiprocessing Services to implement separate paths of execution for tasks that are reentrant and can therefore can be preemptively scheduled.

Using Thread Manager routines, you can create threads and thread pools and set them up to run; turn scheduling on and off; work with stacks; create dialog boxes that yield control to other threads; pass information between threads; install custom scheduling and context-switching functions; and use threads to make asynchronous I/O calls.

The Thread Manager provides only cooperative threading for PowerPC applications. Applications can use the Multiprocessing Services API to create preemptively scheduled tasks.

Note that several Thread Manager functions that did not require you to pass universal procedure pointers (UPPs) for callbacks now require them in Carbon. See the Carbon Porting Notes for more information.

Functions by Task

Creating and Disposing of Threads

`DisposeThread` (page 2091)
Deletes a thread when it finishes executing.

NewThread (page 2104)
Creates a new thread or allocates one from the existing pool of threads.

Creating and Getting Information About Thread Pools

CreateThreadPool (page 2088)
Creates a pool of threads for your application.

GetDefaultThreadStackSize (page 2095)
Determines the default stack size required by a thread.

GetFreeThreadCount (page 2095) Deprecated in Mac OS X v10.3
Determines how many threads are available to be allocated in a thread pool. (Deprecated. There is no replacement.)

GetSpecificFreeThreadCount (page 2096) Deprecated in Mac OS X v10.3
Determines how many threads with a stack size equal to or greater than the specified size are available to be allocated in a thread pool. (Deprecated. There is no replacement.)

Getting Information About Specific Threads

GetCurrentThread (page 2094)
Obtains the thread ID of the currently executing thread.

GetThreadState (page 2098)
Obtains the state of a thread.

ThreadCurrentStackSpace (page 2117)
Determines the amount of stack space that is available for any thread in your application.

Getting Information and Scheduling Threads During Interrupts

GetThreadCurrentTaskRef (page 2097)
Obtains a thread task reference.

GetThreadStateGivenTaskRef (page 2099)
Obtains the state of a thread when your application is not necessarily the current process—for example, during execution of an interrupt function.

SetThreadReadyGivenTaskRef (page 2110)
Changes the state of a thread from stopped to ready when your application is not the current process.

Installing Custom Scheduling, Switching, Terminating, and Debugging Functions

SetDebuggerNotificationProcs (page 2108)
Installs functions that notify the debugger when a thread is created, disposed of, or scheduled.

SetThreadScheduler (page 2110)
Installs a custom scheduling function (custom scheduler).

SetThreadSwitcher (page 2114)
Installs a custom context-switching function for any thread.

SetThreadTerminator (page 2115)
Installs a custom thread-termination function for a thread.

Preventing Scheduling

SetThreadStateEndCritical (page 2113)
Changes the state of the current thread and exits that thread's critical section at the same time.

ThreadBeginCritical (page 2116)
Indicates that the thread is entering a critical code section.

ThreadEndCritical (page 2118)
Indicates that the thread is leaving a critical code section.

Scheduling Threads

SetThreadState (page 2112)
Changes the state of any thread.

YieldToAnyThread (page 2118)
Relinquishes the current thread's control.

YieldToThread (page 2119)
Relinquishes the current thread's control to a particular thread.

Miscellaneous

DisposeDebuggerDisposeThreadUPP (page 2090)

DisposeDebuggerNewThreadUPP (page 2090)

DisposeDebuggerThreadSchedulerUPP (page 2091)

DisposeThreadEntryUPP (page 2092)

DisposeThreadSchedulerUPP (page 2093)

DisposeThreadSwitchUPP (page 2093)

DisposeThreadTerminationUPP (page 2094)

InvokeDebuggerDisposeThreadUPP (page 2100)

InvokeDebuggerNewThreadUPP (page 2100)

Functions

CreateThreadPool

Creates a pool of threads for your application.

```
OSErr CreateThreadPool (
   ThreadStyle threadStyle,
   SInt16 numToCreate,
   Size stackSize
);
```

Parameters

threadStyle

The type of thread to create for this set of threads in the pool. Cooperative is the only type that you can specify. Historically, the Thread Manger supported two types of threads, preemptive and cooperative. However, due to severe limitations on their use, the Thread Manager no longer supports preemptive threads.

numToCreate

The number of threads to create for the pool.

stackSize

The stack size for this set of threads in the pool. This stack must be large enough to handle saved thread context, normal application stack usage, interrupt handling functions, and CPU exceptions. Specify a stack size of 0 to request the Thread Manager's default stack size for the specified type of thread.

Return Value
A result code. See "Thread Manager Result Codes" (page 2134).

Discussion
The `CreateThreadPool` function creates the specified number of threads with the specified stack requirements. It places the threads that it creates into a pool for use by your application.

When you call `CreateThreadPool`, if the Thread Manager is unable to create all the threads that you specify, it does not create any at all and returns the `memFullErr` result code.

The threads in the pool are indistinguishable except by stack size. That is, you cannot refer to them individually. When you want to use a thread to execute some code in your application, you allocate a thread of a specific size from the pool using the `NewThread` function. The `NewThread` function assigns a thread ID to the thread and specifies the function that is the entry point to the thread.

Note that it is not strictly necessary to create a pool of threads before allocating a thread. If you wish, you can use the `NewThread` function to create and allocate a thread in one step. The advantage of using `CreateThreadPool` is that you can allocate memory for threads early in your application's execution before memory is used or fragmented.

Before making any calls to `CreateThreadPool`, be certain that you first have called the Memory Manager function `MaxApplZone` to extend the application heap to its limit. You must call `MaxApplZone` from the main application thread before any other threads in your application run.

To allocate a thread from the pool created with `CreateThreadPool`, use the `NewThread` (page 2104) function.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

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

Declared In
`Threads.h`

DisposeDebuggerDisposeThreadUPP

```
void DisposeDebuggerDisposeThreadUPP (
   DebuggerDisposeThreadUPP userUPP
);
```

Parameters

userUPP

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

DisposeDebuggerNewThreadUPP

```
void DisposeDebuggerNewThreadUPP (
   DebuggerNewThreadUPP userUPP
);
```

Parameters

userUPP

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

DisposeDebuggerThreadSchedulerUPP

```
void DisposeDebuggerThreadSchedulerUPP (
   DebuggerThreadSchedulerUPP userUPP
);
```

Parameters

userUPP

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

DisposeThread

Deletes a thread when it finishes executing.

```
OSErr DisposeThread (
   ThreadID threadToDump,
   void *threadResult,
   Boolean recycleThread
);
```

Parameters

threadToDump

The thread ID of the thread to delete.

threadResult

A pointer to the thread's result. The `DisposeThread` function places this result to an address which you originally specify with the `threadResult` parameter of the `NewThread` function when you create or allocate the thread. Pass a value of `NULL` if you are not interested in obtaining a function result.

recycleThread

A Boolean value that specifies whether to return the thread to the allocation pool or to remove it entirely. Specify `False` to dispose of the thread entirely and `True` to return it to the thread pool.

Return Value

A result code. See "Thread Manager Result Codes" (page 2134).

Discussion

When a thread finishes executing, the Thread Manager automatically calls `DisposeThread` to delete it. Therefore, the only reason for you to explicitly call `DisposeThread` is to recycle a terminating thread. To do so, set the `recycleThread` parameter to `True`. The Thread Manager clears out the thread's internal data structure, resets it, and puts the thread in the thread pool where it can be used again as necessary.

The `DisposeThread` function sets the `threadResult` parameter to the thread's function result. You allocate the storage for the thread result when you create or allocate a thread with the `NewThread` function.

You cannot explicitly dispose of the main application thread. If you attempt to do so, `DisposeThread` returns the `threadProtocolErr` result code.

When your application terminates, the Thread Manager calls `DisposeThread` to terminate any active threads. It terminates stopped and ready threads first but in no special order. It terminates the currently running thread last. This thread should always be the main application thread.

To install a callback function to do special cleanup when a thread terminates, use the `SetThreadTerminator` (page 2115) function.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

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

Declared In
`Threads.h`

DisposeThreadEntryUPP

```
void DisposeThreadEntryUPP (
   ThreadEntryUPP userUPP
);
```

Parameters

userUPP

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

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

Declared In
`Threads.h`

DisposeThreadSchedulerUPP

```
void DisposeThreadSchedulerUPP (
   ThreadSchedulerUPP userUPP
);
```

Parameters

userUPP

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

DisposeThreadSwitchUPP

```
void DisposeThreadSwitchUPP (
   ThreadSwitchUPP userUPP
);
```

Parameters

userUPP

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

DisposeThreadTerminationUPP

```
void DisposeThreadTerminationUPP (
   ThreadTerminationUPP userUPP
);
```

Parameters

userUPP

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

GetCurrentThread

Obtains the thread ID of the currently executing thread.

```
OSErr GetCurrentThread (
    ThreadID * currentThreadID
);
```

Parameters

currentThreadID

On return, a pointer to the thread ID of the current thread.

Return Value

A result code. See "Thread Manager Result Codes" (page 2134).

Discussion

You can use the thread ID obtained by `GetCurrentThread` in functions such as `GetThreadState` and `SetThreadState` to get and set the state of a thread.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

GetDefaultThreadStackSize

Determines the default stack size required by a thread.

```
OSErr GetDefaultThreadStackSize (
   ThreadStyle threadStyle,
   Size *stackSize
);
```

Parameters

threadStyle

The type of thread to get information about. Cooperative is the only type that you can specify. Historically, the Thread Manger supported two types of threads, preemptive and cooperative, but the Thread Manager no longer supports preemptive threads.

stackSize

On return, a pointer to the default stack size (in bytes). When you create a thread pool or an individual thread, this is the stack size that the Thread Manager allocates when you specify the default size.

Return Value

A result code. See "Thread Manager Result Codes" (page 2134).

Discussion

Keep in mind that the default stack size is not an absolute value that you must use but is a rough estimate.

To determine how much stack space is available for a particular thread, use the `ThreadCurrentStackSpace` (page 2117) function.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

GetFreeThreadCount

Determines how many threads are available to be allocated in a thread pool. (Deprecated in Mac OS X v10.3. There is no replacement.)

```
OSErr GetFreeThreadCount (
   ThreadStyle threadStyle,
   SInt16 *freeCount
);
```

Parameters

threadStyle

The type of thread to get information about. Cooperative is the only type that you can specify. Historically, the Thread Manger supported two types of threads, preemptive and cooperative, but the Thread Manager no longer supports preemptive threads.

freeCount
On return, a pointer to the number of threads available to be allocated.

Return Value
A result code. See "Thread Manager Result Codes" (page 2134).

Discussion
The number of threads in the pool varies throughout execution of your application. Calls to `CreateThreadPool` add threads to the pool and calls to the function `NewThread` (page 2104) , when an existing thread is allocated, reduce the number of threads. You also add threads to the pool when you dispose of a thread with the `DisposeThread` (page 2091) function and specify that the thread be recycled.

Use the `GetSpecificFreeThreadCount` (page 2096) function to determine how many threads of a particular stack size are available.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

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
`Threads.h`

GetSpecificFreeThreadCount

Determines how many threads with a stack size equal to or greater than the specified size are available to be allocated in a thread pool. (Deprecated in Mac OS X v10.3. There is no replacement.)

```
OSErr GetSpecificFreeThreadCount (
   ThreadStyle threadStyle,
   Size stackSize,
   SInt16 *freeCount
);
```

Parameters

threadStyle
The type of thread to get information about. Cooperative is the only type that you can specify. Historically, the Thread Manger supported two types of threads, preemptive and cooperative, but the Thread Manager no longer supports preemptive threads.

stackSize
The stack size of the threads to get information about.

freeCount
On return, a pointer to the number of threads of the specified stack size available to be allocated.

Return Value
A result code. See "Thread Manager Result Codes" (page 2134).

Discussion
The `GetSpecificFreeThreadCount` function determines how many threads with a stack size equal to or greater than the specified size are available to be allocated. Use this function instead of `GetFreeThreadCount` (page 2095) when you are interested not simply in the total number of available threads but when you want to know the number of available threads of a specified stack size as well.

The number of threads in the pool varies throughout execution of your application. Calls to the function `CreateThreadPool` (page 2088) add threads to the pool and calls to the function `NewThread` (page 2104) , when an existing thread is allocated, reduce the number of threads. You also add threads to the pool when you dispose of a thread with the `DisposeThread` (page 2091) function and specify that the thread be recycled.

To determine how many threads of any stack size are available, use the `GetFreeThreadCount` function.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

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
`Threads.h`

GetThreadCurrentTaskRef

Obtains a thread task reference.

```
OSErr GetThreadCurrentTaskRef (
   ThreadTaskRef *threadTRef
);
```

Parameters

threadTRef
On return, a pointer to a thread task reference.

Return Value
A result code. See "Thread Manager Result Codes" (page 2134).

Discussion
The thread task reference is somewhat of a misnomer because it identifies your application context, not a particular thread. Identifying your application context is necessary in situations where you aren't guaranteed that your application is the current context—such as during the execution of an interrupt function. In such cases, you need both the thread ID to identify the thread and the thread task reference to identify the application context.

After you obtain the thread task reference, you can use it in the `GetThreadStateGivenTaskRef` (page 2099) and `SetThreadReadyGivenTaskRef` (page 2110) functions to get and set information about specific threads in your application at times when you are not guaranteed that your application is the current context.

To get information about a thread when your application is not the current process, use the `GetThreadStateGivenTaskRef` function.

To change the state of a thread from stopped to ready when your application is not the current process, use the `SetThreadReadyGivenTaskRef` function.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

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

Declared In
`Threads.h`

GetThreadState

Obtains the state of a thread.

```
OSErr GetThreadState (
   ThreadID threadToGet,
   ThreadState *threadState
);
```

Parameters

threadToGet
The thread ID of the thread about which you want information.

threadState
On return, a pointer to the state of the thread specified by `threadToGet`.

Return Value
A result code. See "Thread Manager Result Codes" (page 2134).

Discussion
A thread can be in one of three states: ready to execute (`kReadyThreadState`), stopped (`kStoppedThreadState`), or executing (`kRunningThreadState`).

To change the state of a specified thread, use `SetThreadState` (page 2112).

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

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

Declared In
`Threads.h`

GetThreadStateGivenTaskRef

Obtains the state of a thread when your application is not necessarily the current process—for example, during execution of an interrupt function.

```
OSErr GetThreadStateGivenTaskRef (
   ThreadTaskRef threadTRef,
   ThreadID threadToGet,
   ThreadState *threadState
);
```

Parameters

threadTRef

The thread task reference of the application containing the thread whose state you want to determine.

threadToGet

The thread ID of the thread whose state you want to determine.

threadState

A pointer to a thread state variable in which the function places the state of the specified thread.

Return Value

A result code. See "Thread Manager Result Codes" (page 2134).

Discussion

You can use `GetThreadStateGivenTaskRef` at times when you aren't guaranteed that your application is the current context, such as during execution of an interrupt function. In such cases you must identify the thread task reference (the application context) as well as the thread ID.

To determine the thread task reference (application context) for your application, use the `GetThreadCurrentTaskRef` (page 2097) function.

To change the state of a thread from stopped to ready when your application is not the current process, use the `SetThreadReadyGivenTaskRef` (page 2110) function.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

InvokeDebuggerDisposeThreadUPP

```
void InvokeDebuggerDisposeThreadUPP (
   ThreadID threadDeleted,
   DebuggerDisposeThreadUPP userUPP
);
```

Parameters

userUPP

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

InvokeDebuggerNewThreadUPP

```
void InvokeDebuggerNewThreadUPP (
   ThreadID threadCreated,
   DebuggerNewThreadUPP userUPP
);
```

Parameters

userUPP

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

InvokeDebuggerThreadSchedulerUPP

```
ThreadID InvokeDebuggerThreadSchedulerUPP (
   SchedulerInfoRecPtr schedulerInfo,
   DebuggerThreadSchedulerUPP userUPP
);
```

Parameters

schedulerInfo

userUPP

Return Value

See the description of the `ThreadID` data type.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

InvokeThreadEntryUPP

```
voidPtr InvokeThreadEntryUPP (
   void *threadParam,
   ThreadEntryUPP userUPP
);
```

Parameters

userUPP

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

InvokeThreadSchedulerUPP

```
ThreadID InvokeThreadSchedulerUPP (
   SchedulerInfoRecPtr schedulerInfo,
   ThreadSchedulerUPP userUPP
);
```

Parameters

schedulerInfo

userUPP

Return Value

See the description of the `ThreadID` data type.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

InvokeThreadSwitchUPP

```
void InvokeThreadSwitchUPP (
   ThreadID threadBeingSwitched,
   void *switchProcParam,
   ThreadSwitchUPP userUPP
);
```

Parameters

userUPP

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

InvokeThreadTerminationUPP

```
void InvokeThreadTerminationUPP (
   ThreadID threadTerminated,
   void *terminationProcParam,
   ThreadTerminationUPP userUPP
);
```

Parameters

userUPP

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

NewDebuggerDisposeThreadUPP

```
DebuggerDisposeThreadUPP NewDebuggerDisposeThreadUPP (
   DebuggerDisposeThreadProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value

See the description of the `DebuggerDisposeThreadUPP` data type.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

NewDebuggerNewThreadUPP

```
DebuggerNewThreadUPP NewDebuggerNewThreadUPP (
   DebuggerNewThreadProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value

See the description of the `DebuggerNewThreadUPP` data type.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

NewDebuggerThreadSchedulerUPP

```
DebuggerThreadSchedulerUPP NewDebuggerThreadSchedulerUPP (
   DebuggerThreadSchedulerProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value

See the description of the `DebuggerThreadSchedulerUPP` data type.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

NewThread

Creates a new thread or allocates one from the existing pool of threads.

Modified

```
OSErr NewThread (
   ThreadStyle threadStyle,
   ThreadEntryTPP threadEntry,
   void *threadParam,
   Size stackSize,
   ThreadOptions options,
   void **threadResult,
   ThreadID *threadMade
);
```

Parameters

threadStyle

The type of thread to create. Cooperative is the only type that you can specify. Historically, the Thread Manger supported two types of threads, preemptive and cooperative, but the Thread Manager no longer supports preemptive threads.

threadEntry

A pointer to the thread entry function.

threadParam

A pointer to a value that the Thread Manager passes as a parameter to the thread entry function. Specify `NULL` if you are passing no information.

stackSize

The stack size (in bytes) to allocate for this thread. This stack must be large enough to handle saved thread context, normal application stack usage, interrupt handling functions, and CPU exceptions. Specify a stack size of 0 (zero) to request the Thread Manager's default stack size.

options

Options that define characteristics of the new thread. See the `Thread Option Constants` (page 2132) data type for details on the options. You sum the options together to create a single `options` parameter.

threadResult

On return, a pointer to the address of a location to hold the function result provided by the `Thread Option Constants` (page 2132) function when the thread terminates. Specify `NULL` for this parameter if you are not interested in the function result.

threadMade

On return, a pointer to the thread ID of the newly created or allocated thread. If there is an error, `threadMade` points to a value of `kNoThreadID`.

Return Value

A result code. See "Thread Manager Result Codes" (page 2134).

Discussion

The `NewThread` function obtains a thread ID that you can use in other Thread Manager functions to identify the thread. If you want to allocate a thread from the pool of threads, specify the `kUsePremadeThread` option of the `options` parameter. Otherwise, `NewThread` creates a new thread.

When you request a thread from the existing pool, the Thread Manager allocates one that best fits your specified stack size. If you specify the `kExactMatchThread` option of the `options` parameter, the Thread Manager allocates a thread whose stack exactly matches your stack-size requirement or, if it can't allocate one because no such thread exists, it returns the `threadTooManyReqsErr` result code.

Before making any calls to `NewThread`, be certain that you first have called the Memory Manager function `MaxApplZone` to extend the application heap to its limit. You must call `MaxApplZone` from the main application thread before any other threads in your application run.

When you call the `NewThread` function, you pass, as the `threadEntry` parameter, a pointer to the name of the entry function to the thread. When the newly created thread runs initially, it begins by executing this function.

You can use the `threadParam` parameter to pass thread-specific information to a newly created or allocated thread. In the data structure pointed to by this parameter, you could place something like A5 information or the address of a window to update. You could also use this parameter to specify a place for a thread's local storage.

Be sure to create the storage for the `threadResult` parameter in a place that is guaranteed to be available when the thread terminates—for example, in an application global variable or in a local variable of the application's main function (the main thread, by definition, cannot be disposed of so it is always available). Do not create the storage in a local variable of a subfunction that completes before the thread terminates or the storage will become invalid.

For Carbon applications, the pointer to your thread entry function must be a universal procedure pointer (UPP).

To dispose of a thread, use the DisposeThread function.

See the description of the `Thread Option Constants` (page 2132) data type for details on the characteristics you can specify in the `options` parameter.

For more information about the thread entry function, see the `ThreadEntryProcPtr` (page 2122) function.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Carbon Porting Notes

For Carbon applications, you must create and pass a universal procedure pointer (UPP) to specify the new thread callback. Use the `NewThreadEntryUPP` (page 2106) and `DisposeThreadEntryUPP` (page 2092) functions to create and remove the UPP.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

NewThreadEntryUPP

```
ThreadEntryUPP NewThreadEntryUPP (
   ThreadEntryProcPtr userRoutine
);
```

Parameters

`userRoutine`

Return Value

See the description of the `ThreadEntryUPP` data type.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

NewThreadSchedulerUPP

```
ThreadSchedulerUPP NewThreadSchedulerUPP (
   ThreadSchedulerProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value

See the description of the `ThreadSchedulerUPP` data type.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

NewThreadSwitchUPP

```
ThreadSwitchUPP NewThreadSwitchUPP (
   ThreadSwitchProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value

See the description of the `ThreadSwitchUPP` data type.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

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

Declared In
`Threads.h`

NewThreadTerminationUPP

```
ThreadTerminationUPP NewThreadTerminationUPP (
   ThreadTerminationProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `ThreadTerminationUPP` data type.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

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

Declared In
`Threads.h`

SetDebuggerNotificationProcs

Installs functions that notify the debugger when a thread is created, disposed of, or scheduled.

Modified

```
OSErr SetDebuggerNotificationProcs (
   DebuggerNewThreadTPP notifyNewThread,
   DebuggerDisposeThreadTPP notifyDisposeThread,
   DebuggerThreadSchedulerTPP notifyThreadScheduler
);
```

Parameters

notifyNewThread

A pointer to the callback function that notifies the debugger when a thread is created.

notifyDisposeThread

A pointer to the callback function that notifies the debugger when a thread is disposed of. This function is called whether you manually dispose of a thread with the `DisposeThread` function or if a thread disposes of itself automatically when it returns from its highest level of code.

notifyThreadScheduler

A pointer to the callback function that notifies the debugger when a thread is scheduled.

Return Value
A result code. See "Thread Manager Result Codes" (page 2134).

Discussion
You generally use this function only during development of an application.

The `SetDebuggerNotificationProcs` function installs three separate callback functions that return the thread ID of a newly created thread, the thread ID of a newly disposed of thread, and the thread ID of a newly scheduled thread.

The `SetDebuggerNotificationProcs` function always installs all three of the debugging functions. You cannot set only one or two of these functions, nor can you chain them together. These restrictions ensure that the function that calls `SetDebuggerNotificationProcs` owns all three of the debugging functions. If you want to prevent one or two of these debugging functions from being called, you can do so by setting them to `NULL`.

To guarantee that the debugger is getting an accurate view of scheduling, the Thread Manager doesn't call the scheduling-notification callback function until both the generic Thread Manager scheduler and any custom thread scheduler have decided on a thread to schedule.

For Carbon applications, the pointers you pass to specify the callbacks must be universal procedure pointers (UPPs).

To create or allocate a new thread, use the `NewThread` (page 2104) function.

To dispose of a thread, use the `DisposeThread` function.

To schedule a thread, you can use a yield function such as `YieldToAnyThread` (page 2118) or `YieldToThread` (page 2119) or a function to change the state of a thread, such as `SetThreadState` (page 2112).

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Carbon Porting Notes

For Carbon applications, you must create and pass a universal procedure pointer (UPP) to specify the notification callbacks. You must use the designated UPP creation and disposal functions. For example, for the new thread notifier, you call the `NewDebuggerNewThreadUPP` (page 2104) and `DisposeDebuggerNewThreadUPP` (page 2090) functions.

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

Declared In
`Threads.h`

SetThreadReadyGivenTaskRef

Changes the state of a thread from stopped to ready when your application is not the current process.

```
OSErr SetThreadReadyGivenTaskRef (
   ThreadTaskRef threadTRef,
   ThreadID threadToSet
);
```

Parameters

threadTRef

The thread task reference of the application containing the thread whose state you want to change.

threadToSet

The thread ID of the thread whose state you want to change.

Return Value

A result code. See "Thread Manager Result Codes" (page 2134).

Discussion

When you mark a thread as ready to run with this function, the Thread Manager does not put it immediately into the scheduling queue but does so the next time it reschedules threads.

You can use `SetThreadStateGivenTaskRef` at times when you aren't guaranteed that your application is the current context, such as during execution of an interrupt function. In such cases you must identify the thread task reference (the application context) as well as the thread ID.

You obtain the thread task reference for your application with the `GetThreadCurrentTaskRef` (page 2097) function.

The `SetThreadReadyGivenTaskRef` function allows you to do one thing only—change a thread from stopped to ready to execute. You cannot change the state of an executing thread to ready or stopped, nor can you change the state of a ready thread to executing or stopped with this call.

To determine the state of a thread when your application is not the current process, use the `GetThreadStateGivenTaskRef` (page 2099) function.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

SetThreadScheduler

Installs a custom scheduling function (custom scheduler).

Modified

```
OSErr SetThreadScheduler (
   ThreadSchedulerTPP threadScheduler
);
```

Parameters

threadScheduler

A pointer to a custom scheduler. Specify `NULL` if you want to remove an installed custom scheduler and use the default Thread Manager scheduling mechanism.

Return Value

A result code. See "Thread Manager Result Codes" (page 2134).

Discussion

The `SetThreadScheduler` function installs a custom scheduler that runs in conjunction with the default Thread Manager scheduling mechanism. The Thread Manager uses a scheduler information structure to pass the custom scheduler the ID of the current thread and the ID of the thread that the Thread Manager has scheduled to run next.

A custom scheduler should return to the Thread Manager the ID of the thread that it determines to schedule. If it does not determine a particular thread to schedule, it should return the constant `kNoThreadID` and the Thread Manager default scheduling mechanism schedules the next thread.

If you already have a custom scheduler installed when you call `SetThreadScheduler`, it replaces the old one with a new one. If you want to remove your custom scheduler and return to using the default Thread Manager scheduling mechanism, call `SetThreadScheduler` and specify a value of `NULL` for the parameter.

The `SetThreadScheduler` function automatically disables scheduling to avoid any reentrancy problems with the custom scheduling function. Therefore, in your custom scheduling function, you should make no yield calls or other calls that would cause scheduling to occur.

For Carbon applications, the pointer to your thread scheduler function must be a universal procedure pointer (UPP).

For more information on the custom scheduling function, see the `ThreadSchedulerProcPtr` (page 2123) function.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Carbon Porting Notes

For Carbon applications, you must create and pass a universal procedure pointer (UPP) to specify the thread scheduler callback. Use the `NewThreadSchedulerUPP` (page 2107) and `DisposeThreadSchedulerUPP` (page 2093) functions to create and remove the UPP.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

SetThreadState

Changes the state of any thread.

```
OSErr SetThreadState (
   ThreadID threadToSet,
   ThreadState newState,
   ThreadID suggestedThread
);
```

Parameters

threadToSet

The thread ID of the thread whose state is to be changed.

newState

The new state for the thread. You can specify ready to execute (`kReadyThreadState`), stopped (`kStoppedThreadState`), or executing (`kRunningThreadState`).

suggestedThread

The thread ID of the next thread to run. You specify this thread if you are changing the state of the currently executing thread to stopped or ready to run. Pass `kNoThreadID` if you do not want to specify a particular thread to run next. In this case, the Thread Manager schedules the next available thread to run.

Return Value

A result code. See "Thread Manager Result Codes" (page 2134).

Discussion

The effect of `SetThreadState` depends on whether the thread you specify for changing is the currently executing thread or another thread. If you specify the current thread and thus change the state to stopped or ready, `SetThreadState` invokes the Thread Manager scheduling mechanism. The current thread relinquishes control (it is put in the state you specify, stopped or ready) and the Thread Manager schedules the thread that you specify with the `suggestedThread` parameter. If this thread is unavailable for running, or if you passed `kNoThreadID`, the Thread Manager schedules the next available thread.

If you change the state of the current thread to ready, the Thread Manager suspends it awaiting of the CPU. When it is rescheduled, `SetThreadState` regains control and returns to the function that called it.

If you have installed a custom scheduler, the Thread Manager passes it the thread ID of the suspended thread.

If you specify a thread other than the currently executing thread, no rescheduling occurs. If you change the state from ready to stopped, the thread is removed from the scheduling queue. The Thread Manager does not schedule this thread for execution again until you change its state to ready. On the other hand, if you change the state from stopped to ready, you have in effect put the thread in the scheduling queue, and the Thread Manager gives it CPU time as soon as it reaches the top of the scheduling queue.

Threads must yield in the CPU addressing mode (24-bit or 32-bit) in which the application was launched.

To obtain the state of any thread, use the `GetThreadState` (page 2098) function.

To relinquish control to the next available thread, use the `YieldToAnyThread` (page 2118) function. To relinquish control to a specific thread, use the `YieldToThread` (page 2119) function.

To set the state of the current thread before it exits a critical section of code, use the `SetThreadStateEndCritical` (page 2113) function.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

SetThreadStateEndCritical

Changes the state of the current thread and exits that thread's critical section at the same time.

```
OSErr SetThreadStateEndCritical (
   ThreadID threadToSet,
   ThreadState newState,
   ThreadID suggestedThread
);
```

Parameters

threadToSet

The thread ID of the thread whose state is to be changed.

newState

The new state for the thread. You can specify ready to execute (`kReadyThreadState`), stopped (`kStoppedThreadState`) or executing (`kRunningThreadState`).

suggestedThread

The thread ID of the next thread to run. You specify this thread if you are changing the state of the currently executing thread to stopped or ready to run. Pass `kNoThreadID` if you do not want to specify a particular thread to run next. In this case, the Thread Manager schedules the next available thread to run.

Return Value

A result code. See "Thread Manager Result Codes" (page 2134).

Discussion

The `SetThreadStateEndCritical` function does in one step the same thing that `ThreadEndCritical` and `SetThreadState` functions do in two steps.

Historically, the primary purpose of the `SetThreadStateEndCritical` function was to close the scheduling window at the end of a critical section. A preemptive thread that was waiting while the critical section of code was executing could begin executing before you changed the state of the current thread to stopped with the `SetThreadState` function. Obviously, because the Thread Manager no longer supports preemptive threads, this function is no longer necessary to close the scheduling window, but you can still use it to change the state of a thread and exit a critical section in one step instead of two.

When you change the state of the currently executing thread, the Thread Manager schedules the thread you specify with the `suggestedThread` parameter. If this thread is unavailable or if you pass `kNoThreadID`, the Thread Manager schedules the next available thread.

To mark a section of code as critical, use the `ThreadBeginCritical` (page 2116) and the `ThreadEndCritical` (page 2118) functions.

To change the state of any thread, use the `SetThreadState` (page 2112) function.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

SetThreadSwitcher

Installs a custom context-switching function for any thread.

Modified

```
OSErr SetThreadSwitcher (
   ThreadID thread,
   ThreadSwitchTPP threadSwitcher,
   void *switchProcParam,
   Boolean inOrOut
);
```

Parameters

thread

The thread ID of the thread to associate with a context-switching function.

threadSwitcher

A pointer to the context-switching function.

switchProcParam

A pointer to a thread-specific parameter that you pass to the context-switching function.

inOrOut

A Boolean value that indicates whether the Thread Manager calls the context-switching function when the specified thread switches in (`True`) or when it is switched out by another thread (`False`).

Return Value

A result code. See "Thread Manager Result Codes" (page 2134).

Discussion

The custom switching function allows you to save context information in addition to the default context information that the Thread Manager automatically saves when it switches contexts. The default context information consists of the CPU registers, the FPU registers (if any), and the location of the thread's context.

You must actually define two context-switching functions, one for leaving a thread and another for entering a thread. When leaving a thread, you call the outer context-switching function to save additional context information. When reentering a thread, you call the inner context-switching function to restore the extra information that was saved on exit. Use the `inOrOut` parameter of the `SetThreadSwitcher` function to specify which type of context-switching function is being installed.

You can pass a different `switchProcParam` parameter to each thread, which allows you to write a single, application-wide custom switching function and then pass any thread-specific information when the Thread Manager calls the switching function for that thread.

The `SetThreadSwitcher` function automatically disables scheduling to avoid any reentrancy problems with the custom switching function. Therefore, in the custom switching function, you should make no yield calls or other calls that would cause scheduling to occur.

For Carbon applications, the pointer to your thread switcher function must be a universal procedure pointer (UPP).

For more information on the custom context-switching function, see the `ThreadSwitchProcPtr` (page 2124) function.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Carbon Porting Notes

For Carbon applications, you must create and pass a universal procedure pointer (UPP) to specify the thread switcher callback. Use the `NewThreadSwitchUPP` (page 2107) and `DisposeThreadSwitchUPP` (page 2093) functions to create and remove the UPP.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

SetThreadTerminator

Installs a custom thread-termination function for a thread.

Modified

```
OSErr SetThreadTerminator (
   ThreadID thread,
   ThreadTerminationTPP threadTerminator,
   void *terminationProcParam
);
```

Parameters

thread

The thread ID of the thread to associate with the thread-termination function.

threadTerminator
A pointer to the thread-termination function.

terminationProcParam
A pointer to a thread-specific parameter that you pass to the thread-termination function.

Return Value
A result code. See "Thread Manager Result Codes" (page 2134).

Discussion
The Thread Manager calls the custom termination function whenever the specified thread completes execution of its code or when you manually dispose of the thread with the `DisposeThread` (page 2091) function.

You can pass a different `terminationProcParam` parameter to each thread, which allows you to write a single, application-wide custom thread-termination function and then pass any thread-specific information when the Thread Manager calls the termination function for that thread.

For Carbon applications, the pointer to your thread terminator function must be a universal procedure pointer (UPP).

For more information on the custom thread-termination function, see the `ThreadTerminationProcPtr` (page 2125) function.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

Carbon Porting Notes

For Carbon applications, you must create and pass a universal procedure pointer (UPP) to specify the thread terminator callback. Use the `NewThreadTerminationUPP` (page 2108) and `DisposeThreadTerminationUPP` (page 2094) functions to create and remove the UPP.

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

Declared In
`Threads.h`

ThreadBeginCritical

Indicates that the thread is entering a critical code section.

```
OSErr ThreadBeginCritical (
   void
);
```

Return Value
A result code. See "Thread Manager Result Codes" (page 2134).

Discussion
The `ThreadBeginCritical` function disables scheduling by marking the beginning of a section of critical code. That is, no other threads in the current application can run—even if the current thread yields control—until the current thread exits the critical section (by calling the `ThreadEndCritical` function). Disabling scheduling allows the currently executing function to look at or change shared or global data safely. You can nest critical sections within a thread.

To mark the end of a critical code section and turn scheduling back on, use the `ThreadEndCritical` (page 2118) function. If you also need to set the state of the current thread before scheduling is turned back on, use the `SetThreadStateEndCritical` (page 2113) function.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

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

Declared In
`Threads.h`

ThreadCurrentStackSpace

Determines the amount of stack space that is available for any thread in your application.

```
OSErr ThreadCurrentStackSpace (
   ThreadID thread,
   ByteCount *freeStack
);
```

Parameters

thread
The thread ID of the thread about which you want information.

freeStack
On return, a pointer to the amount of stack space (in bytes) that is available to the specified thread.

Return Value
A result code. See "Thread Manager Result Codes" (page 2134).

Discussion
This function is primarily useful during debugging since you determine the maximum amount of stack space you need for any particular thread before you ship your application. However, if your application calls a recursive function that could call itself many times, you might want to use `ThreadCurrentStackSpace` to keep track of the stack space and take appropriate action if it becomes too low.

To determine the default size that the Thread Manager assigns to threads use the `GetDefaultThreadStackSize` (page 2095) function.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

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

Declared In
`Threads.h`

ThreadEndCritical

Indicates that the thread is leaving a critical code section.

```
OSErr ThreadEndCritical (
   void
);
```

Return Value
A result code. See "Thread Manager Result Codes" (page 2134).

Discussion
After a call to the Thread, all scheduling operations are now available to the application.

Use the `ThreadBeginCritical` (page 2116) function to mark the beginning of a critical code section and turn scheduling off.

If you need to set the state of the current thread before scheduling is turned back on, use the `SetThreadStateEndCritical` (page 2113) function.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

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

Declared In
`Threads.h`

YieldToAnyThread

Relinquishes the current thread's control.

```
OSErr YieldToAnyThread (
   void
);
```

Return Value
A result code. See "Thread Manager Result Codes" (page 2134).

Discussion
The `YieldToAnyThread` function invokes the Thread Manager's scheduling mechanism. The current thread relinquishes control and the Thread Manager schedules the next available thread.

The current thread is suspended in the ready state and awaits rescheduling when the CPU is available. When the suspended thread is scheduled again, `YieldToAnyThread` regains control and returns to the function that called it.

If you have installed a custom scheduler, the Thread Manager passes it the thread ID of the suspended thread.

In each thread you must make one or more strategically placed calls to relinquish control to another thread. You can either make this yield call or another yield call such as `YieldToThread`; or you can make a call such as `SetThreadState` to explicitly change the state of the thread.

Threads must yield in the CPU addressing mode (24-bit or 32-bit) in which the application was launched.

To relinquish control to a specific thread, use the `YieldToThread` (page 2119) function.

To change the state of a specified thread, use the `SetThreadState` (page 2112) function.

Special Considerations

Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

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

Declared In
`Threads.h`

YieldToThread

Relinquishes the current thread's control to a particular thread.

```
OSErr YieldToThread (
   ThreadID suggestedThread
);
```

Parameters

`suggestedThread`
The ID of the thread to yield control to.

Return Value
A result code. See "Thread Manager Result Codes" (page 2134).

Discussion
The `YieldToThread` function invokes the Thread Manager's scheduling mechanism. The current thread relinquishes control and passes the thread ID of a thread for the Thread Manager to schedule. The Thread Manager schedules this thread if it is available. Otherwise, the Thread Manager schedules the next available thread.

The current thread is suspended in the ready state and awaits rescheduling when the CPU is available. When the suspended thread is scheduled again, `YieldToThread` regains control and returns to the function that called it.

If you have installed a custom scheduler, the Thread Manager passes it the thread ID of the suspended thread.

In each thread you must make one or more strategically placed calls to relinquish control to another thread. You can either make this yield call or another yield call such as `YieldToAnyThread`; or you can make a call such as `SetThreadState` to explicitly change the state of the thread.

Threads must yield in the CPU addressing mode (24-bit or 32-bit) in which the application was launched.

To relinquish control without naming a specific thread, use the `YieldToAnyThread` (page 2118) function.

To change the state of a specified thread, use the `SetThreadState` (page 2112) function.

Special Considerations
Active development with the Thread Manager is not recommended. The API is intended only for developers who are porting their applications to Mac OS X and whose code relies on the cooperative threading model. If you are writing a new Carbon application, you should use POSIX threads or the Multiprocessing Services API instead. See *Threading Programming Guide* for more information.

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

Declared In
`Threads.h`

Callbacks

DebuggerDisposeThreadProcPtr

Defines a pointer to a dispose thread debugging callback function. A dispose thread debugging callback function is a debugging function that the Thread Manager calls whenever it disposes of a thread.

```
typedef void (*DebuggerDisposeThreadProcPtr)
(
    ThreadID threadDeleted
);
```

If you name your function `MyDebuggerDisposeThreadProc`, you would declare it like this:

```
void MyDebuggerDisposeThreadProcPtr (
    ThreadID threadDeleted
);
```

Parameters

threadDeleted

The thread ID of the thread being disposed of.

Return Value

Discussion

The `MyDebuggerDisposeThreadCallback` function is one of three debugging functions that you can install with the `SetDebuggerNotificationProcs` (page 2108) function. The Thread Manager calls `MyDebuggerDisposeThreadCallback` whenever an application disposes of a thread. The thread manager calls this debugging function whether you manually call `DisposeThread` (page 2091) to dispose of a thread or if a thread finishes executing its code and the Thread Manager automatically disposes of it.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

DebuggerNewThreadProcPtr

Defines a pointer to a new thread debugging callback function. A new thread debugging callback function is a debugging function that the Thread Manager calls whenever it creates a new thread.

```
typedef void (*DebuggerNewThreadProcPtr)
(
    ThreadID threadCreated
);
```

If you name your function `MyDebuggerNewThreadProc`, you would declare it like this:

```
void MyDebuggerNewThreadProcPtr (
    ThreadID threadCreated
);
```

Parameters

threadCreated

The thread ID of the thread being created.

Return Value

Discussion

The `MyDebuggerNewThreadCallback` function is one of three debugging functions that you can install with the `SetDebuggerNotificationProcs` (page 2108) function. The Thread Manager calls `MyDebuggerNewThreadCallback` whenever an application creates or allocates a new thread with the `NewThread` (page 2104) function. The Thread Manager does not call `MyDebuggerNewThreadCallback` when an application creates a thread pool with the `CreateThreadPool` function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

DebuggerThreadSchedulerProcPtr

Defines a pointer to a thread scheduler debugging callback function. A thread scheduler debugging callback function is a debugging function that the Thread Manager calls whenever a thread is scheduled.

```
typedef ThreadID (*DebuggerThreadSchedulerProcPtr)
(
    SchedulerInfoRecPtr schedulerInfo
);
```

If you name your function `MyDebuggerThreadSchedulerProc`, you would declare it like this:

```
ThreadID MyDebuggerThreadSchedulerProcPtr
(
    SchedulerInfoRecPtr schedulerInfo
);
```

Parameters

schedulerInfo

A pointer to a scheduler information structure that the `SetDebuggerNotificationProcs` function passes to the `MyDebuggerThreadSchedulerCallback` function. Among other information, the scheduler information structure contains the ID of the current thread and the ID of the thread that the Thread Manager has scheduled to run next.

Return Value

See the description of the `ThreadID` data type.

Discussion

The `MyDebuggerThreadSchedulerCallback` function is one of three debugging functions that you can install with the `SetDebuggerNotificationProcs` (page 2108) function. The Thread Manager calls `MyDebuggerThreadSchedulerCallback` whenever an application schedules a new thread to run. The `MyDebuggerThreadSchedulerCallback` function gets the last look at the thread being scheduled—that is, the Thread Manager calls this function after the Thread Manager default scheduling mechanism and a custom scheduler, if you have installed one, decide on the next thread to schedule.

If you wish, you can use this debugging callback function to schedule a different thread than that chosen by the Thread Manager and any custom scheduling function. The `MyDebuggerThreadSchedulerCallback` returns the thread ID of the next thread to schedule. The `MyDebuggerThreadSchedulerCallback` can specify `kNoThreadID` for the thread ID if you do not want to change the decision of the Thread Manager default scheduler or a custom scheduler.

To schedule a thread, use functions such as `YieldToAnyThread` (page 2118) , `YieldToThread` (page 2119) , and `SetThreadState` (page 2112).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

ThreadEntryProcPtr

Defines a pointer to a thread entry callback function. Your thread entry callback function provides an entry point to a thread that you create in your application.

```
typedef voidPtr (*ThreadEntryProcPtr)
(
    void * threadParam
);
```

If you name your function `MyThreadEntryProc`, you would declare it like this:

```
voidPtr MyThreadEntryProcPtr (
    void * threadParam
);
```

Parameters

threadParam

A pointer to a `void` data structure passed to this function by the `NewThread` function.

Return Value

Discussion

When you create or allocate a new thread with the `NewThread` function, you pass the name of this entry function. You also pass a parameter that the Thread Manager passes on to the `MyThreadEntryCallback` function. You can use this parameter to pass thread-specific information to the newly created or allocated thread. For example, you could pass something like A5 information or the address of a window to update. Or you could use this parameter to specify local storage for a thread that other threads could access.

When the code in a thread finishes executing, the Thread Manager automatically calls the `DisposeThread` (page 2091) function to dispose of the thread. The `MyThreadEntryCallback` function passes its function result to `DisposeThread`. The `DisposeThread` function passes this result back to the `NewThread` function that called `MyThreadEntryCallback` to begin with.

This mechanism allows you to spawn a thread that does some work and then continue with your original thread. When the spawned thread is finished doing its work—for example a calculation—it returns the result to the original thread.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

ThreadSchedulerProcPtr

Defines a pointer to a thread scheduler callback function. Your thread scheduler callback function supplements the Thread Manager default scheduling mechanism.

```
typedef ThreadID (*ThreadSchedulerProcPtr)
(
    SchedulerInfoRecPtr schedulerInfo
);
```

If you name your function `MyThreadSchedulerProc`, you would declare it like this:

```
ThreadID MyThreadSchedulerProcPtr (
    SchedulerInfoRecPtr schedulerInfo
);
```

Parameters

schedulerInfo

A pointer to the scheduler information structure that the Thread Manager uses to pass information to `MyThreadSchedulerCallback`.

Return Value

See the description of the `ThreadID` data type.

Discussion

The `MyThreadSchedulerCallback` function does not supplant the Thread Manager scheduling mechanism but rather works in conjunction with it.

Whenever scheduling occurs, the Thread Manager passes a scheduler information structure to `MyThreadSchedulerCallback`. Among other information, the scheduler information structure contains the thread ID of the current thread and the thread ID of the thread that the application has scheduled to run next.

The `MyThreadSchedulerCallback` function returns to the Thread Manager the thread ID of the thread that it has chosen to schedule and the Thread Manager does the actual scheduling. If `MyThreadSchedulerCallback` decides not to schedule a thread, it returns the constant `kNoThreadID` and the Thread Manager default scheduling mechanism schedules the next thread.

When the `SetThreadScheduler` function installs the custom scheduler, it automatically disables scheduling to avoid any reentrancy problems. Therefore, in the custom scheduler, you should make no yield calls or other calls that would cause scheduling to occur.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

ThreadSwitchProcPtr

Defines a pointer to a thread switch callback function. Your thread switch callback function adds to the thread context information that the Thread Manager saves and restores.

```
typedef void (*ThreadSwitchProcPtr) (
    ThreadID threadBeingSwitched,
    void * switchProcParam
);
```

If you name your function `MyThreadSwitchProc`, you would declare it like this:

```
void MyThreadSwitchProcPtr (
    ThreadID threadBeingSwitched,
    void * switchProcParam
);
```

Parameters

threadBeingSwitched

The thread ID of the thread whose context is being switched.

switchProcParam

A pointer to a parameter that the `SetThreadSwitcher` function passes to `MyThreadSwitchCallback`.

Return Value

Discussion

The custom switching function allows you to save and restore context information in addition to the default context information that the Thread Manager automatically saves and restores when it switches contexts. You must actually define two context-switching functions, one for leaving a thread and another for entering a thread. When leaving a thread, you call the outer context-switching function to save additional context information. When reentering a thread, you call the inner context-switching function to restore the extra information that was saved on exit.

The default context information consists of the CPU registers, the FPU registers (if any), and the location of the thread's context.

When the `SetThreadSwitcher` function installs the custom switching function, it automatically disables scheduling to avoid any reentrancy problems. Therefore, in the custom switching function, you should make no yield calls or other calls that would cause scheduling to occur.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

ThreadTerminationProcPtr

Defines a pointer to a thread termination callback function. Your thread termination callback function does additional cleanup when the code in a thread finishes executing.

```
typedef void (*ThreadTerminationProcPtr)
(
    ThreadID threadTerminated,
    void * terminationProcParam
);
```

If you name your function `MyThreadTerminationProc`, you would declare it like this:

```
void MyThreadTerminationProcPtr (
    ThreadID threadTerminated,
    void * terminationProcParam
);
```

Parameters

threadTerminated

The thread ID of the thread being disposed of.

terminationProcParam

A pointer to a `void` data structure that the `SetThreadTerminator` function passes to `MyThreadTerminationCallback`.

Return Value

Discussion
You use the `SetThreadTerminator` function to install the `MyThreadTerminationCallback` custom termination function. The custom termination function allows you to do additional cleanup when the code in a thread finishes executing or when you call the `DisposeThread` (page 2091) function to manually dispose of a thread.

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

Declared In
`Threads.h`

Data Types

DebuggerDisposeThreadUPP

```
typedef DebuggerDisposeThreadProcPtr DebuggerDisposeThreadUPP;
```

Discussion
For more information, see the description of the DebuggerDisposeThreadUPP () callback function.

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

Declared In
`Threads.h`

DebuggerDisposeThreadTPP

```
typedef DebuggerDisposeThreadUPP DebuggerDisposeThreadTPP;
```

Discussion

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

Declared In
`Threads.h`

DebuggerNewThreadTPP

```
typedef DebuggerNewThreadUPP DebuggerNewThreadTPP;
```

Discussion

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

Declared In
Threads.h

DebuggerNewThreadUPP

```
typedef DebuggerNewThreadProcPtr DebuggerNewThreadUPP;
```

Discussion
For more information, see the description of the DebuggerNewThreadUPP () callback function.

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

Declared In
Threads.h

DebuggerThreadSchedulerUPP

```
typedef DebuggerThreadSchedulerProcPtr DebuggerThreadSchedulerUPP;
```

Discussion
For more information, see the description of the DebuggerThreadSchedulerUPP () callback function.

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

Declared In
Threads.h

DebuggerThreadSchedulerTPP

```
typedef DebuggerThreadSchedulerUPP DebuggerThreadSchedulerTPP;
```

Discussion

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

Declared In
Threads.h

SchedulerInfoRec

```
struct SchedulerInfoRec {
    UInt32 InfoRecSize;
    ThreadID CurrentThreadID;
    ThreadID SuggestedThreadID;
    ThreadID InterruptedCoopThreadID;
};
typedef struct SchedulerInfoRec SchedulerInfoRec;
typedef SchedulerInfoRec * SchedulerInfoRecPtr;
```

Fields

`InfoRecSize`
The size of the structure.

`CurrentThreadID`
The thread ID of the current thread.

`SuggestedThreadID`
The thread ID of the thread that the application has suggested to run.

`InterruptedCoopThreadID`
Historically, the thread ID of a preempted cooperative thread if a cooperative thread has been interrupted and has not yet resumed execution. Because it no longer supports preemptive threads, the Thread Manager always passes the constant `kNoThreadID` to indicate that there is no thread that has been interrupted.

Discussion

You can, if you wish, use the `SetThreadScheduler` (page 2110) function to install a custom scheduling function to work in conjunction with the default Thread Manager scheduling mechanism. The Thread Manager uses the scheduler information structure to pass information to the custom scheduling function that allows it to decide which thread, if any, to schedule next.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

ThreadEntryTPP

```
typedef ThreadEntryUPP ThreadEntryTPP;
```

Discussion

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Threads.h`

ThreadEntryUPP

```
typedef ThreadEntryProcPtr ThreadEntryUPP;
```

Discussion
For more information, see the description of the ThreadEntryUPP () callback function.

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

Declared In
`Threads.h`

ThreadSchedulerTPP

```
typedef ThreadSchedulerUPP ThreadSchedulerTPP;
```

Discussion

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

Declared In
`Threads.h`

ThreadSchedulerUPP

```
typedef ThreadSchedulerProcPtr ThreadSchedulerUPP;
```

Discussion
For more information, see the description of the ThreadSchedulerUPP () callback function.

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

Declared In
`Threads.h`

ThreadSwitchTPP

```
typedef ThreadSwitchUPP ThreadSwitchTPP;
```

Discussion

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

Declared In
`Threads.h`

ThreadSwitchUPP

```
typedef ThreadSwitchProcPtr ThreadSwitchUPP;
```

Discussion
For more information, see the description of the ThreadSwitchUPP () callback function.

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

Declared In
`Threads.h`

ThreadTaskRef

Represents a thread task reference.

```
typedef void* ThreadTaskRef;
```

Discussion
In certain cases, such as during execution of an interrupt function, your application is not guaranteed to be the current process. Since threads are defined within an application context, it follows that in cases such as these, you cannot get or set information about any particular threads in your application unless you have a way of identifying the application context. The thread task reference gives you a way of doing this.

You can obtain the thread task reference by calling `GetThreadCurrentTaskRef` (page 2097) at a time when you know your application is the current context. Later, during execution of an interrupt function, you can use the thread task reference to identify your application. For example, you can pass the thread task reference to functions such as `GetThreadStateGivenTaskRef` (page 2099) and `SetThreadReadyGivenTaskRef` (page 2110) in an interrupt function to get and set information about the state of particular threads in your application.

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

Declared In
`Threads.h`

ThreadTerminationTPP

```
typedef ThreadTerminationUPP ThreadTerminationTPP;
```

Discussion

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

Declared In
`Threads.h`

ThreadTerminationUPP

```
typedef ThreadTerminationProcPtr ThreadTerminationUPP;
```

Discussion
For more information, see the description of the ThreadTerminationUPP () callback function.

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

Declared In
`Threads.h`

Constants

Thread ID Constants

The `ThreadID` data type defines the thread ID.

```
typedef UInt32 ThreadID;
enum {
    kNoThreadID = 0,
    kCurrentThreadID = 1,
    kApplicationThreadID = 2
};
```

Constants

`kNoThreadID`
Indicates no thread; for example, you can use a function such as `SetThreadState` (page 2112) to put the current thread in the stopped state and pass `kNoThreadID` to indicate that you don't care which thread runs next.

Available in Mac OS X v10.0 and later.

Declared in `Threads.h`.

`kCurrentThreadID`
Identifies the currently executing thread.

Available in Mac OS X v10.0 and later.

Declared in `Threads.h`.

`kApplicationThreadID`
Identifies the main application thread this is the cooperative thread that the Thread Manager creates at launch time. You cannot dispose of this thread. All applications—even those that are not aware of the Thread Manager—have one main application thread. The Thread Manager assumes that the main application thread is responsible for event gathering when an operating-system event occurs, the Thread Manager schedules the main application thread as the next thread to execute.

Available in Mac OS X v10.0 and later.

Declared in `Threads.h`.

Discussion
The Thread Manager assigns a thread ID to each thread that you create or allocate with the `NewThread` (page 2104) function. The thread ID uniquely identifies a thread within an application context. You can use the thread ID in functions that schedule execution of a particular thread, dispose of a thread, and get and set information about a thread; for example, you pass the thread ID to functions such as `YieldToThread` (page 2119) , `DisposeThread` (page 2091) , and `GetThreadState` (page 2098) .

In addition to the specific thread IDs that the `NewThread` function returns, you can use the three Thread Manager constants described here.

Thread Option Constants

```
typedef UInt32 ThreadOptions;
enum {
    kNewSuspend = (1 << 0),
    kUsePremadeThread = (1 << 1),
    kCreateIfNeeded = (1 << 2),
    kFPUNotNeeded = (1 << 3),
    kExactMatchThread = (1 << 4)
};
```

Constants

`kNewSuspend`
Begin a new thread in the stopped state.
Available in Mac OS X v10.0 and later.
Declared in `Threads.h`.

`kUsePremadeThread`
Use a thread from the existing supply.
Available in Mac OS X v10.0 and later.
Declared in `Threads.h`.

`kCreateIfNeeded`
Create a new thread if one with the proper style and stack size requirements does not exist.
Available in Mac OS X v10.0 and later.
Declared in `Threads.h`.

`kFPUNotNeeded`
Do not save the FPU context. This saves time when switching contexts. Note, however, that for PowerPC threads, the Thread Manager always saves the FPU registers regardless of how you set this option. Because the PowerPC microprocessor uses the FPU registers for optimizations, they could contain necessary information.
Available in Mac OS X v10.0 and later.
Declared in `Threads.h`.

`kExactMatchThread`
Allocate a thread from the pool only if it exactly matches the stack-size request. Without this option, a thread is allocated that best fits the request—that is, a thread whose stack is greater than or equal to the requested size.
Available in Mac OS X v10.0 and later.
Declared in `Threads.h`.

Discussion
When you create or allocate a new thread with the `NewThread` (page 2104) function, you can specify thread options that define certain characteristics of the thread, using the values described here. To specify more than one option, you sum them together and pass them as a single parameter to the `NewThread` function.

The `ThreadOptions` data type defines the thread options.

Thread State Constants

```
typedef UInt16 ThreadState;
enum {
    kReadyThreadState = 0,
    kStoppedThreadState = 1,
    kRunningThreadState = 2
};
```

Constants
`kReadyThreadState`
The thread is ready to run.
Available in Mac OS X v10.0 and later.
Declared in `Threads.h`.

`kStoppedThreadState`
The thread is stopped and not ready to run.
Available in Mac OS X v10.0 and later.
Declared in `Threads.h`.

`kRunningThreadState`
The thread is running.
Available in Mac OS X v10.0 and later.
Declared in `Threads.h`.

Discussion
The Thread Manager functions which get and set information about the state of a thread, such as `GetThreadState` (page 2098) and `SetThreadState` (page 2112) , use these values.

Thread Style Constants

```
typedef UInt32 ThreadStyle;
enum {
    kCooperativeThread = 1L << 0,
    kPreemptiveThread = 1L << 1
};
```

Constants
`kCooperativeThread`
Available in Mac OS X v10.0 and later.
Declared in `Threads.h`.

`kPreemptiveThread`
Available in Mac OS X v10.0 and later.
Declared in `Threads.h`.

Discussion
Historically, the Thread Manager defined two types of threads to run in an application context: cooperative and preemptive, but now it supports only cooperative threads.

Although the Thread Manager only supports a single type of thread, many Thread Manager functions (for historical reasons) require you to use the thread type to specify the type of the thread.

The `ThreadStyle` data type specifies the type of a thread.

Because there is only one type of thread (cooperative) the thread type accepts a single value, `kCooperativeThread`.

Result Codes

The most common result codes returned by Thread Manager are listed below.

Result Code	Value	Description
`threadTooManyReqsErr`	-617	Available in Mac OS X v10.0 and later.
`threadNotFoundErr`	-618	Available in Mac OS X v10.0 and later.
`threadProtocolErr`	-619	Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availability information by using the Thread Manager selectors defined in the Gestalt Manager. For more information see Inside Mac OS X: Gestalt Manager Reference.

Time Manager Reference

Framework: CoreServices/CoreServices.h

Declared in Timer.h

Overview

The Time Manager allows applications and other software to schedule routines for execution at a later time. By suitably defining the routine that is to be executed later, you can use the Time Manager to accomplish a wide range of time-related activities. For example, because a routine can reschedule itself for later execution, the Time Manager allows your application to perform periodic or repeated actions. You can use the Time Manager to schedule routines for execution after a specified delay; set up tasks that run periodically; compute the time a routine takes to run; and coordinate and synchronize actions in the Macintosh computer.

The Time Manager provides a hardware-independent method of performing these time-related tasks. In general, you should use the Time Manager instead of timing loops, which can vary in duration because they depend on clock speed and interrupt-handling speed.

Carbon supports the Time Manager. However, the interface for callbacks will change because the current task record is accessible only from 68K code.

Functions by Task

Installing and Removing Tasks

`InstallTimeTask` (page 2137) Deprecated in Mac OS X v10.4
: Installs a task structure into the Time Manager task queue. (Deprecated. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

`InstallXTimeTask` (page 2138) Deprecated in Mac OS X v10.4
: Installs a task, taking advantage of the drift-free, fixed-frequency timing services of the extended Time Manager. (Deprecated. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

`RemoveTimeTask` (page 2143) Deprecated in Mac OS X v10.4
: Removes a task from the Time Manager queue. (Deprecated. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

Activating Tasks

`PrimeTimeTask` (page 2142) Deprecated in Mac OS X v10.4
Activates a task in the Time Manager queue. (Deprecated. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

Measuring Time

`Microseconds` (page 2141)
Determines the number of microseconds that have elapsed since system startup time.

Working With Your Time Manager Callback Function

`NewTimerUPP` (page 2141)
Creates a new universal procedure pointer (UPP) to your Time Manager task callback. (Deprecated. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

`InvokeTimerUPP` (page 2140)
Invokes your Time Manager task callback function. (Deprecated. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

`DisposeTimerUPP` (page 2137)
Disposes of the universal procedure pointer (UPP) to your Time Manager task callback function. (Deprecated. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

Obsolete Functions

`InsTime` (page 2139) Deprecated in Mac OS X v10.4
Installs a task record into the Time Manager task queue. (Deprecated. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

`InsXTime` (page 2139) Deprecated in Mac OS X v10.4
Installs an extended task record into the Time Manager task queue. (Deprecated. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

`PrimeTime` (page 2142) Deprecated in Mac OS X v10.4
Activates a task in the Time Manager queue. (Deprecated. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

`RmvTime` (page 2144) Deprecated in Mac OS X v10.4
Remove a task from the Time Manager queue. (Deprecated. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

Functions

DisposeTimerUPP

Disposes of the universal procedure pointer (UPP) to your Time Manager task callback function. (**Deprecated.** Use Carbon Event Loop timers or Cocoa NSTImers instead.)

```
void DisposeTimerUPP (
   TimerUPP userUPP
);
```

Parameters

userUPP

A UPP to your callback function.

Discussion

See the callback `TimerProcPtr` (page 2145) for more information.

Special Considerations

Carbon Event timers and Cocoa NSTimers provide a simpler and more efficient way to handle timed or periodic tasks. For more information about using Carbon Event timers, see the timers section in *Carbon Event Manager Programming Guide*. For information about NSTimers, see *Timer Programming Topics for Cocoa*. Both Carbon event timers and NSTimers are built on top of the lower-level Core Foundation CFRunLoop timers. For CFRunLoop information, see *Run Loops*.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Declared In

`Timer.h`

InstallTimeTask

Installs a task structure into the Time Manager task queue. (Deprecated in Mac OS X v10.4. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

```
OSErr InstallTimeTask (
   QElemPtr tmTaskPtr
);
```

Parameters

tmTaskPtr

A pointer to an original task structure to be installed in the queue.

Return Value

A result code. See "Time Manager Result Codes" (page 2147).

Discussion

The `InstallTimeTask` function adds the Time Manager task structure specified by the `tmTaskPtr` parameter to the Time Manager queue. Your application should fill in the `tmAddr` field of the task structure and should set the `tmCount` field to 0. The `tmTaskPtr` parameter must point to an original Time Manager task structure.

With the revised and extended Time Managers, you can set the `tmAddr` field to `NULL` if you do not want a task to execute when the delay passed to the `PrimeTime` function expires. Also, the revised Time Manager resets the high-order bit of the `qType` field to 0 when you call the `InsTime` function.

The `InstallTimeTask` function, which returns a value of type `OSErr`, takes the place of `InsTime`.

Special Considerations

Carbon Event timers and Cocoa NSTimers provide a simpler and more efficient way to handle timed or periodic tasks. For more information about using Carbon Event timers, see the timers section in *Carbon Event Manager Programming Guide*. For information about NSTimers, see *Timer Programming Topics for Cocoa*. Both Carbon event timers and NSTimers are built on top of the lower-level Core Foundation CFRunLoop timers. For CFRunLoop information, see *Run Loops*.

Availability

Available in CarbonLib 1.0.2 and later when running Mac OS 9.1 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`Timer.h`

InstallXTimeTask

Installs a task, taking advantage of the drift-free, fixed-frequency timing services of the extended Time Manager. (Deprecated in Mac OS X v10.4. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

```
OSErr InstallXTimeTask (
   QElemPtr tmTaskPtr
);
```

Parameters

`tmTaskPtr`

A pointer to an extended task structure to be installed in the queue.

Return Value

A result code. See "Time Manager Result Codes" (page 2147).

Discussion

The `InstallXTimeTask` function adds the Time Manager task structure specified by `tmTaskPtr` to the Time Manager queue. Use `InstallXTimeTask` only if you wish to use the drift-free, fixed-frequency timing services of the extended Time Manager; use `InstallTimeTask` in all other cases. The `tmTaskPtr` parameter must point to an extended Time Manager task structure. Your application must fill in the `tmAddr` field of that task. You should set the `tmWakeUp` and `tmReserved` fields to 0 the first time you call `InsXTime`.

With the extended Time Manager, you can set `tmAddr` to `NULL` if you do not want a task to execute when the delay passed to `PrimeTime` expires. Also, `InsXTime` resets the high-order bit of the `qType` field to 0.

The `InstallXTimeTask` function, which returns a value of type `OSErr`, takes the place of InsXTime.

Special Considerations

Carbon Event timers and Cocoa NSTimers provide a simpler and more efficient way to handle timed or periodic tasks. For more information about using Carbon Event timers, see the timers section in *Carbon Event Manager Programming Guide*. For information about NSTimers, see *Timer Programming Topics for Cocoa*. Both Carbon event timers and NSTimers are built on top of the lower-level Core Foundation CFRunLoop timers. For CFRunLoop information, see *Run Loops*.

Availability

Available in CarbonLib 1.0.2 and later when running Mac OS 9.1 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`Timer.h`

InsTime

Installs a task record into the Time Manager task queue. (Deprecated in Mac OS X v10.4. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

Not Recommended

```
void InsTime (
   QElemPtr tmTaskPtr
);
```

Special Considerations

Carbon Event timers and Cocoa NSTimers provide a simpler and more efficient way to handle timed or periodic tasks. For more information about using Carbon Event timers, see the timers section in *Carbon Event Manager Programming Guide*. For information about NSTimers, see *Timer Programming Topics for Cocoa*. Both Carbon event timers and NSTimers are built on top of the lower-level Core Foundation CFRunLoop timers. For CFRunLoop information, see *Run Loops*.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`Timer.h`

InsXTime

Installs an extended task record into the Time Manager task queue. (Deprecated in Mac OS X v10.4. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

Not Recommended

```
void InsXTime (
   QElemPtr tmTaskPtr
);
```

Special Considerations

Carbon Event timers and Cocoa NSTimers provide a simpler and more efficient way to handle timed or periodic tasks. For more information about using Carbon Event timers, see the timers section in *Carbon Event Manager Programming Guide*. For information about NSTimers, see *Timer Programming Topics for Cocoa*. Both Carbon event timers and NSTimers are built on top of the lower-level Core Foundation CFRunLoop timers. For CFRunLoop information, see *Run Loops*.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`Timer.h`

InvokeTimerUPP

Invokes your Time Manager task callback function. (**Deprecated.** Use Carbon Event Loop timers or Cocoa NSTImers instead.)

```
void InvokeTimerUPP (
   TMTaskPtr tmTaskPtr,
   TimerUPP userUPP
);
```

Parameters

tmTaskPtr
A pointer to a structure of type `TMTask` containing the information about the task.

userUPP
A UPP to your callback function.

Discussion
See the callback `TimerProcPtr` (page 2145) for more information.

Special Considerations

Carbon Event timers and Cocoa NSTimers provide a simpler and more efficient way to handle timed or periodic tasks. For more information about using Carbon Event timers, see the timers section in *Carbon Event Manager Programming Guide*. For information about NSTimers, see *Timer Programming Topics for Cocoa*. Both Carbon event timers and NSTimers are built on top of the lower-level Core Foundation CFRunLoop timers. For CFRunLoop information, see *Run Loops*.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`Timer.h`

Microseconds

Determines the number of microseconds that have elapsed since system startup time.

```
void Microseconds (
   UnsignedWide *microTickCount
);
```

Parameters

microTickCount

The number of microseconds elapsed since system startup.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X 10.0 and later.

Declared In

`Timer.h`

NewTimerUPP

Creates a new universal procedure pointer (UPP) to your Time Manager task callback. (**Deprecated.** Use Carbon Event Loop timers or Cocoa NSTImers instead.)

```
TimerUPP NewTimerUPP (
   TimerProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your Time Manager event callback function. For information on how to create a Time Manager event callback see `TimerProcPtr` (page 2145)

Return Value

A UPP to your callback function. See the description of the `TimerUPP` data type.

Discussion

See the callback `TimerProcPtr` (page 2145) for more information.

Special Considerations

Carbon Event timers and Cocoa NSTimers provide a simpler and more efficient way to handle timed or periodic tasks. For more information about using Carbon Event timers, see the timers section in *Carbon Event Manager Programming Guide*. For information about NSTimers, see *Timer Programming Topics for Cocoa*. Both Carbon event timers and NSTimers are built on top of the lower-level Core Foundation CFRunLoop timers. For CFRunLoop information, see *Run Loops*.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Declared In

`Timer.h`

PrimeTime

Activates a task in the Time Manager queue. (Deprecated in Mac OS X v10.4. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

Not Recommended

```
void PrimeTime (
   QElemPtr tmTaskPtr,
   long count
);
```

Discussion
This function is deprecated. You should use the function `PrimeTimeTask` (page 2142) instead.

Special Considerations

Carbon Event timers and Cocoa NSTimers provide a simpler and more efficient way to handle timed or periodic tasks. For more information about using Carbon Event timers, see the timers section in *Carbon Event Manager Programming Guide*. For information about NSTimers, see *Timer Programming Topics for Cocoa*. Both Carbon event timers and NSTimers are built on top of the lower-level Core Foundation CFRunLoop timers. For CFRunLoop information, see *Run Loops*.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`Timer.h`

PrimeTimeTask

Activates a task in the Time Manager queue. (Deprecated in Mac OS X v10.4. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

```
OSErr PrimeTimeTask (
   QElemPtr tmTaskPtr,
   long count
);
```

Parameters

tmTaskPtr
A pointer to a task structure already installed in the queue.

count
The desired delay before execution of the task.

Return Value
A result code. See "Time Manager Result Codes" (page 2147).

Discussion
The `PrimeTimeTask` function schedules the task specified by the `tmAddr` field of the structure pointed to by the *tmTaskPtr* parameter for execution after the delay specified by the *count* parameter has elapsed.

If the *count* parameter is a positive value, it is interpreted as milliseconds. If *count* is a negative value, it is interpreted in negated microseconds. Microsecond delays are allowable only in the revised and extended Time Managers.

The task record specified by the *tmTaskPtr* parameter must already be installed in the queue (by a previous call to the functions `InstallTimeTask` (page 2137) or `InstallXTimeTask` (page 2138)) before your application calls the `PrimeTimeTask` function. The `PrimeTimeTask` function returns immediately, and the specified task is executed after the specified delay has elapsed. If you call the `PrimeTimeTask` function with a time delay of 0, the task runs as soon as interrupts are enabled.

In the revised and extended Time Managers, the `PrimeTimeTask` function sets the high-order bit of the `qType` field to 1. In addition, any value of the *count* parameter that exceeds the maximum millisecond delay is reduced to the maximum. If you stop an unexpired task (by calling the function `RemoveTimeTask` (page 2143)) and then reinstall it (by calling the `InstallXTimeTask` function), you can continue the previous delay by calling the `PrimeTimeTask` function with the *count* parameter set to 0.

Special Considerations

Carbon Event timers and Cocoa NSTimers provide a simpler and more efficient way to handle timed or periodic tasks. For more information about using Carbon Event timers, see the timers section in *Carbon Event Manager Programming Guide*. For information about NSTimers, see *Timer Programming Topics for Cocoa*. Both Carbon event timers and NSTimers are built on top of the lower-level Core Foundation CFRunLoop timers. For CFRunLoop information, see *Run Loops*.

Availability

Available in CarbonLib 1.0.2 and later when running Mac OS 9.1 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`Timer.h`

RemoveTimeTask

Removes a task from the Time Manager queue. (Deprecated in Mac OS X v10.4. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

```
OSErr RemoveTimeTask (
   QElemPtr tmTaskPtr
);
```

Parameters

tmTaskPtr

A pointer to a task structure to be removed from the queue.

Return Value

A result code. See "Time Manager Result Codes" (page 2147).

Discussion

The `RemoveTimeTask` function removes the Time Manager task structure specified by the *tmTaskPtr* parameter from the Time Manager queue. In both the revised and extended Time Managers, if the specified task record is active (that is, if it has been activated but the specified time has not yet elapsed), the `tmCount` field of the task structure returns the amount of time remaining. To provide the greatest accuracy, the unused

time is reported as negated microseconds if that value is small enough to fit into the `tmCount` field (even if the delay was originally specified in milliseconds); otherwise, the unused time is reported in positive milliseconds. If the time has already expired, the `tmCount` field contains 0.

In the revised and extended Time Managers, the `RemoveTimeTask` function sets the high-order bit of the `qType` field to 0.

Special Considerations

Carbon Event timers and Cocoa NSTimers provide a simpler and more efficient way to handle timed or periodic tasks. For more information about using Carbon Event timers, see the timers section in *Carbon Event Manager Programming Guide*. For information about NSTimers, see *Timer Programming Topics for Cocoa*. Both Carbon event timers and NSTimers are built on top of the lower-level Core Foundation CFRunLoop timers. For CFRunLoop information, see *Run Loops*.

Availability
Available in CarbonLib 1.0.2 and later when running Mac OS 9.1 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`Timer.h`

RmvTime

Remove a task from the Time Manager queue. (Deprecated in Mac OS X v10.4. Use Carbon Event Loop timers or Cocoa NSTImers instead.)

Not Recommended

```
void RmvTime (
   QElemPtr tmTaskPtr
);
```

Special Considerations

Carbon Event timers and Cocoa NSTimers provide a simpler and more efficient way to handle timed or periodic tasks. For more information about using Carbon Event timers, see the timers section in *Carbon Event Manager Programming Guide*. For information about NSTimers, see *Timer Programming Topics for Cocoa*. Both Carbon event timers and NSTimers are built on top of the lower-level Core Foundation CFRunLoop timers. For CFRunLoop information, see *Run Loops*.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`Timer.h`

Callbacks

TimerProcPtr

Defines a pointer to your application-defined Time Manager task that is executed after a specified delay.

```
typedef void (*TimerProcPtr) (
    TMTaskPtr tmTaskPtr
);
```

If you name your function `MyTimerProc`, you would declare it like this:

```
void MyTimerProc (
    TMTaskPtr tmTaskPtr
);
```

Parameters

tmTaskPtr

A pointer to a structure of type `TMTask` containing the information about the task.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Timer.h`

Data Types

TimerUPP

Defines a data type for the `TimerProcPtr` callback function.

```
typedef TimerProcPtr TimerUPP;
```

Discussion

For more information, see the description of the `TimerProcPtr` (page 2145) callback function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Timer.h`

TMTask

Contains information for a Time Manager task.

```
struct TMTask {
    QElemPtr qLink;
    short qType;
    TimerUPP tmAddr;
    long tmCount;
    long tmWakeUp;
    long tmReserved;
};
typedef struct TMTask TMTask;
typedef TMTask * TMTaskPtr;
```

Fields

`qLink`

A pointer to the next element in the Time Manager queue. This field is used internally by the Time Manager.

`qType`

The type of queue. The Time Manager automatically sets this field to the appropriate value. The high-order bit of this field is a flag that indicates whether the task is active.

`tmAddr`

A pointer to the function that is to execute after the delay specified in a call to `PrimeTime`.

`tmCount`

Reserved in the original Time Manager. In the revised or extended Time Manager, the amount of time remaining until the task's scheduled execution time. This field is valid only after you call `RmvTime` with a task that has not yet executed.

`tmWakeUp`

In the extended Time Manager, the time when the task specified in the `tmAddr` field was last executed. This field is used internally by the Time Manager. You should set it to 0 when you first install a task structure.

`tmReserved`

Reserved.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Timer.h`

Constants

Active Task Constant

Defines a constant for an active task.

```
enum {
    kTMTaskActive = (1L << 15)
};
```

Constants

`kTMTaskActive`

The high bit of the `qType` field in the `TMTask` structure is set if the task is active.

Available in Mac OS X v10.0 and later.

Declared in `Timer.h`.

Result Codes

The most common result codes returned by Time Manager is `noErr`, which has a value of 0.

Gestalt Constants

You can check for version and feature availability information by using the Time Manager Version selectors defined in the Gestalt Manager. For more information see *Inside Mac OS X: Gestalt Manager Reference*.

Unicode Utilities Reference

Framework: CoreServices/CoreServices.h
Declared in UnicodeUtilities.h

Overview

Unicode Utilities allow applications and text service components (such as input methods) to perform various operations on Unicode text; for example, Unicode key translation. Resources defined for use with Unicode Utilities permit control of Unicode-related text behavior, such as the specification of Unicode keyboard layouts.

Carbon fully supports the Unicode Utilities.

Functions by Task

Inputting Unicode Text

`UCKeyTranslate` (page 2162)
Converts a combination of a virtual key code, a modifier key state, and a dead-key state into a string of one or more Unicode characters.

Comparing Unicode Strings

`UCCreateCollator` (page 2155)
Creates an object encapsulating locale and collation information, for the purpose of performing Unicode string comparison.

`UCCompareText` (page 2151)
Uses locale-specific collation information to compare Unicode strings.

`UCGetCollationKey` (page 2160)
Uses locale-specific collation information to generate a collation key for a Unicode string.

`UCCompareCollationKeys` (page 2150)
Uses collation keys to compare Unicode strings.

`UCDisposeCollator` (page 2158)
Disposes a collator object.

`UCCompareTextDefault` (page 2153)
Uses the default system locale to compare Unicode strings.

`UCCompareTextNoLocale` (page 2154)
Uses a fixed, locale-insensitive order to compare Unicode strings.

Identifying Unicode Text Boundaries

`UCCreateTextBreakLocator` (page 2156)
Creates an object encapsulating locale and text-break information, for the purpose of finding boundaries in Unicode text.

`UCFindTextBreak` (page 2159)
Uses locale-specific text-break information to find boundaries in Unicode text.

`UCDisposeTextBreakLocator` (page 2158)
Disposes a text-break locator object.

Functions

UCCompareCollationKeys

Uses collation keys to compare Unicode strings.

```
OSStatus UCCompareCollationKeys (
   const UCCollationValue *key1Ptr,
   ItemCount key1Length,
   const UCCollationValue *key2Ptr,
   ItemCount key2Length,
   Boolean *equivalent,
   SInt32 *order
);
```

Parameters

key1Ptr
A pointer to the collation key (a `UCCollationValue` array) for the first string to compare. You can obtain a collation key with the function `UCGetCollationKey` (page 2160). The collation key supplied in `key1Ptr` for the first string must be generated with the same collator object as that used to generate the collation key supplied in `key2Ptr` for the second string.

key1Length
An `ItemCount` value specifying the actual length of the collation key supplied in the `key1Ptr` parameter. You can obtain this value from the function `UCGetCollationKey` (page 2160) when you obtain the new collation key.

key2Ptr
A pointer to the collation key (a `UCCollationValue` array) for the second string to compare. You can obtain a collation key with the function `UCGetCollationKey` (page 2160). The collation key supplied in `key2Ptr` for the second string must be generated with the same collator object as that used to generate the collation key supplied in `key1Ptr` for the first string.

key2Length
An `ItemCount` value specifying the actual length of the collation key supplied in the `key2Ptr` parameter. You can obtain this value from the function `UCGetCollationKey` (page 2160) when you obtain the new collation key.

equivalent

A pointer to a `Boolean` value or pass `NULL`. On return, `UCCompareCollationKeys` produces a value of `true` if the strings represented by the collation keys are equivalent for the options you have specified in the collator object. If you wish simply to sort a list of strings in order, using your specified options, you can pass `NULL` for the `equivalent` parameter and only use the `order` parameter's result. In this case, all available comparison criteria are used to put the strings in a deterministic order, even if they are considered "equivalent" for the options you have specified. Note that you can set either the `equivalent` or the `order` parameters to `NULL`, but not both.

order

A pointer to a signed, 32-bit integer value, or pass `NULL`. If you wish simply to test the strings represented by the collation keys for equivalence, using your specified options (which can be much faster than determining ordering), you can pass `NULL` for the `order` parameter and only use the `equivalent` parameter's result. (Note that either the `equivalent` or the `order` parameters may be `NULL`, but not both.

Return Value
A result code. This function can return `paramErr`, for example, if `key1Ptr` or `key2Ptr` are `NULL`.

Discussion
If you wish to compare the same strings several times, as when sorting a list of strings, it may be most efficient for you to derive a collation key for each string and then compare the collation keys. A collation key is a transformation of the string that depends on the collator object (that is, it depends on the locale, the collation variant if any, and the collation options).

Collation keys that are generated using the same collator object—but for different strings—can quickly be compared with each other, without further reference to the collator object or collation tables. The disadvantage is that the collation keys may be rather large. After you use the function `UCGetCollationKey` (page 2160) to create a collation key from a given string and collator object, you can call the `UCCompareCollationKeys` function to compare two collation keys that were generated with the same collator object.

If you are comparing different strings, it may be more efficient for you to call the function `UCCompareText` (page 2151) multiple times using the same collator object.

Note that collation keys should be used only in a runtime context. They should not be stored in a persistent state (such as to disk) because the format of a collation key could change in the future.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeUtilities.h`

UCCompareText

Uses locale-specific collation information to compare Unicode strings.

```
OSStatus UCCompareText (
   CollatorRef collatorRef,
   const UniChar *text1Ptr,
   UniCharCount text1Length,
   const UniChar *text2Ptr,
   UniCharCount text2Length,
   Boolean *equivalent,
   SInt32 *order
);
```

Parameters

collatorRef

A valid reference to a collator object; `NULL` is not allowed. You can use the function `UCCreateCollator` (page 2155) to obtain a collator reference.

text1Ptr

A pointer to the first Unicode string (a `UniChar` array) to compare.

text1Length

The total count of Unicode characters in the first string being compared.

text2Ptr

A pointer to the second Unicode string to compare.

text2Length

The total count of Unicode characters in the second string being compared.

equivalent

A pointer to a `Boolean` value or `NULL`. On return, `UCCompareText` produces a value of `true` if the strings are equivalent for the options you have specified in the collator object. If you wish simply to sort a list of strings in order, using your specified options, you can pass `NULL` for the `equivalent` parameter and only use the `order` parameter's result. In this case, all available comparison criteria are used to put the strings in a deterministic order, even if they are considered "equivalent" for the options you have specified. Note that you can set either the `equivalent` or the `order` parameters to `NULL`, but not both.

order

A pointer to a signed, 32-bit integer value, or pass `NULL`. If you wish simply to test strings for equivalence, using your specified options (which can be much faster than determining ordering), you can pass `NULL` for the `order` parameter and only use the `equivalent` parameter's result. (Note that either the `equivalent` or the `order` parameters may be `NULL`, but not both.

Return Value

A result code. The function can return `paramErr` (for example, if `collatorRef`, `text1Ptr`, or `text2Ptr` are `NULL`.

Discussion

You can use the `UCCompareText` function to perform various types of string comparison for a given set of locale and collation specifications. You can

- simply test whether two strings are equivalent
- determine the relative ordering of two strings
- check whether a given string is equivalent to any string in an ordered list

You can also call the `UCCompareText` function multiple times to compare different strings using the same collator object. If you wish to compare the same strings several times, as when sorting a list of strings, it may be more efficient for you to derive a collation key for each string and then compare the collation keys. For more on comparison using collation keys, see the functions `UCGetCollationKey` (page 2160) and `UCCompareCollationKeys` (page 2150).

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeUtilities.h`

UCCompareTextDefault

Uses the default system locale to compare Unicode strings.

```
OSStatus UCCompareTextDefault (
   UCCollateOptions options,
   const UniChar *text1Ptr,
   UniCharCount text1Length,
   const UniChar *text2Ptr,
   UniCharCount text2Length,
   Boolean *equivalent,
   SInt32 *order
);
```

Parameters

options
A `UCCollateOptions` value specifying any collation options for the string comparison.

text1Ptr
A pointer to the first Unicode string (a `UniChar` array) to compare.

text1Length
The total count of Unicode characters in the first string being compared.

text2Ptr
A pointer to the second Unicode string to compare.

text2Length
The total count of Unicode characters in the second string being compared.

equivalent
A pointer to a `Boolean` value or pass `NULL`. On return, `UCCompareTextDefault` produces a value of `true` if the strings are equivalent for the options you have specified. If you wish simply to sort a list of strings in order, using your specified options, you can pass `NULL` for the `equivalent` parameter and only use the `order` parameter's result. In this case, all available comparison criteria are used to put the strings in a deterministic order, even if they are considered "equivalent" for the options you have specified. Note that you can set either the `equivalent` or the `order` parameters to `NULL`, but not both.

order

A pointer to a signed, 32-bit integer value, or pass `NULL`. If you wish simply to test the strings for equivalence, using your specified options (which can be much faster than determining ordering), you can pass `NULL` for the `order` parameter and only use the `equivalent` parameter's result. (Note that either the `equivalent` or the `order` parameters may be `NULL`, but not both.

Return Value
A result code.

Discussion
You can call the `UCCompareTextDefault` function when you want to use a simple collation function that requires minimum setup. This function uses the system default collation order (that is, the collation order for a `LocaleRef` of `NULL` and a variant of 0), and it does not require a collator object or collation keys.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeUtilities.h`

UCCompareTextNoLocale

Uses a fixed, locale-insensitive order to compare Unicode strings.

```
OSStatus UCCompareTextNoLocale (
   UCCollateOptions options,
   const UniChar *text1Ptr,
   UniCharCount text1Length,
   const UniChar *text2Ptr,
   UniCharCount text2Length,
   Boolean *equivalent,
   SInt32 *order
);
```

Parameters

options

A `UCCollateOptions` value specifying the fixed ordering scheme to use for the string comparison. This value must be nonzero. Bits 24-31 of `UCCollateOptionsValue` specify which fixed ordering scheme to use. Currently there is only scheme—`kUCCollateTypeHFSExtended`. See "Fixed Ordering Scheme" (page 2177) for additional details.

text1Ptr

A pointer to the first Unicode string (a `UniChar` array) to compare.

text1Length

The total count of Unicode characters in the first string being compared.

text2Ptr

A pointer to the second Unicode string to compare.

text2Length

The total count of Unicode characters in the second string being compared.

equivalent

A pointer to a `Boolean` value or pass `NULL`. On return, `UCCompareTextNoLocale` produces a value of `true` if the strings are equivalent for the ordering scheme you have specified. If you wish simply to sort a list of strings in order, using the specified ordering scheme, you can pass `NULL` for the `equivalent` parameter and only use the `order` parameter's result. In this case, all available comparison criteria are used to put the strings in a deterministic order, even if they are considered "equivalent" for the specified ordering scheme. Note that you can set either the `equivalent` or the `order` parameters to `NULL`, but not both.

order

A pointer to a signed, 32-bit integer value, or pass `NULL`. If you wish simply to test the strings for equivalence, using the specified ordering scheme (which can be much faster than determining ordering), you can pass `NULL` for the `order` parameter and only use the `equivalent` parameter's result. (Note that either the `equivalent` or the `order` parameters may be `NULL`, but not both.

Return Value
A result code. This function can return `paramErr` if you pass an invalid value for one of the parameters. For example, if you pass `0` for the `options` paramter, the function returns `paramErr`.

Discussion
You can call the `UCCompareTextNoLocale` function when you want to perform a fixed, locale-insensitive comparison that is guaranteed not to change from one system release to the next. This type of comparison could be used for sorting a Unicode key string in a database, for example. The `UCCompareTextNoLocale` function can provide comparison according to various fixed ordering schemes (only one is supported for Mac OS 8.6 and 9.0). This type of comparison is not usually used for a user-visible ordering, so the ordering schemes need not match any user's expectation of a sensible collation order.

The `UCCompareTextNoLocale` function does not require a collator object or collation keys. Another advantage of `UCCompareTextNoLocale` on Mac OS 9 is that it is exported from the `UnicodeUtilitiesCoreLib` library, which does not depend on other libraries (the other comparison functions exported from `UnicodeUtilitiesLib`, which depends on `LocalesLib` and `TextCommon`).

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeUtilities.h`

UCCreateCollator

Creates an object encapsulating locale and collation information, for the purpose of performing Unicode string comparison.

```
OSStatus UCCreateCollator (
   LocaleRef locale,
   LocaleOperationVariant opVariant,
   UCCollateOptions options,
   CollatorRef *collatorRef
);
```

Parameters

locale

A valid `LocaleRef` representing a specific locale, or pass `NULL` to request the default system locale. You can supply the value `kUnicodeCollationClass` in the `opClass` parameter of the Locales Utilities functions `LocaleOperationCountLocales` and `LocaleOperationGetLocales` to obtain the locales available for collation on the current system.

opVariant

A `LocaleOperationVariant` value identifying a collation variant within the locale specified in the `locale` parameter. You can also pass 0 to request the default collation variant for any locale. To obtain the varieties of locale-specific collation that are currently available, you can supply the value `kUnicodeCollationClass` in the `opClass` parameter of the Locales Utilities functions `LocaleOperationCountLocales` and `LocaleOperationGetLocales`.

options

A `UCCollateOptions` value specifying any collation options that you want to use for the string comparison.

collatorRef

A pointer to a value of type `CollatorRef`. On return, the `CollatorRef` value contains a valid reference to a new collator object.

Return Value

A result code. The function can return memory errors and `paramErr`, for example, if the `collatorRef` parameter is `NULL`. It can also return resource errors in Mac OS 9 and CarbonLib.

Discussion

To perform Unicode string comparison, you must supply locale and collation specifications to a collation function such as `UCCompareText` (page 2151). You provide this information by means of a collator object, created via the `UCCreateCollator` function. When finished with the collator object, you dispose of it using the function `UCDisposeCollator` (page 2158).

Special Considerations

The collator object is allocated in the current heap. This function can move memory.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.

Available in Mac OS X 10.0 and later.

Declared In

`UnicodeUtilities.h`

UCCreateTextBreakLocator

Creates an object encapsulating locale and text-break information, for the purpose of finding boundaries in Unicode text.

```
OSStatus UCCreateTextBreakLocator (
   LocaleRef locale,
   LocaleOperationVariant opVariant,
   UCTextBreakType breakTypes,
   TextBreakLocatorRef *breakRef
);
```

Parameters

locale

A valid `LocaleRef` representing a specific locale, or pass `NULL` to request the default system locale. You can supply the value `kUnicodeTextBreakClass` in the `opClass` parameter of the Locales Utilities functions `LocaleOperationCountLocales` and `LocaleOperationGetLocales` to obtain the locales available for finding text boundaries on the current system.

opVariant

A `LocaleOperationVariant` value identifying a text-break operation variant within the locale specified in the `locale` parameter. You can also pass 0 to request the default text-break variant for any locale. To obtain the varieties of locale-specific text-break variants that are currently available, you can supply the value `kUnicodeTextBreakClass` in the `opClass` parameter of the Locales Utilities functions `LocaleOperationCountLocales` and `LocaleOperationGetLocales`.

breakTypes

A `UCTextBreakType` value specifying each type of text boundary that the text-break locator should support. You do not need to create a text-break locator solely for the `BreakChar` type; it is locale-independent and automatically supported by the function `UCFindTextBreak` (page 2159). If `BreakChar` is the only type for which you call the `UCCreateTextBreakLocator` function, on return the `breakRef` parameter returns a `NULL` value (with no error).

breakRef

A pointer to a value of type `TextBreakLocatorRef`. On return, the `TextBreakLocatorRef` value contains a valid reference to a new text-break locator object.

Return Value

A result code. The function can return memory errors and `paramErr` (for example, if the `breakRef` parameter is `NULL` or if invalid bits are set in the `breakTypes` parameter). It can also return resource errors in Mac OS 9 and CarbonLib.

Discussion

To find boundaries in Unicode text, you must supply locale and text-break specifications to the function `UCFindTextBreak` (page 2159). You provide this information by means of a text-break locator object, created via the `UCCreateTextBreakLocator` function. When finished with the text-break locator object, you should dispose of it using the function `UCDisposeTextBreakLocator` (page 2158).

The `UCCreateTextBreakLocator` function creates a text-break locator object for a specified locale, a specified text-break variant within that locale, and a specified set of break types. The different types of breaks or boundaries in a line of Unicode text can include

- Boundaries of characters (treating surrogate pairs as a single character).
- Boundaries of character clusters. A cluster is a group of characters that should be treated as single text element for editing operations such as cursor movement. Typically this includes groups such as a base character followed by a sequence of combining characters, for example, a Hangul syllable represented as a sequence of conjoining jamo characters or an Indic consonant cluster.
- Boundaries of words. This can be used to determine what to highlight as the result of a double-click.
- Potential line break locations.

Special Considerations

This function can move memory.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.

Declared In

`UnicodeUtilities.h`

UCDisposeCollator

Disposes a collator object.

```
OSStatus UCDisposeCollator (
   CollatorRef *collatorRef
);
```

Parameters

`collatorRef`

A reference to a valid collator object. The `UCDisposeCollator` function sets `*collatorRef` to `NULL`.

Return Value

A result code.

Discussion

To perform Unicode string comparison, you must supply locale and collation specifications to a collation function such as `UCCompareText` (page 2151). You provide this information by means of a collator object, created via the function `UCCreateCollator` (page 2155). When finished with the collator object, you should dispose of it using the function `UCDisposeCollator`.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.
Available in Mac OS X 10.0 and later.

Declared In

`UnicodeUtilities.h`

UCDisposeTextBreakLocator

Disposes a text-break locator object.

```
OSStatus UCDisposeTextBreakLocator (
   TextBreakLocatorRef *breakRef
);
```

Parameters

`breakRef`

A reference to a valid text-break locator object. The `UCDisposeTextBreakLocator` function sets `*breakRef` to `NULL`.

Return Value

A result code. This function can return `paramErr`, for example, if the `breakRef` parameter is `NULL`.

Discussion
To find boundaries in Unicode text, you must supply locale and text-break specifications to the function `UCFindTextBreak` (page 2159). You provide this information by means of a text-break locator object, created via the function `UCCreateTextBreakLocator` (page 2156). When finished with the text-break locator object, you should dispose of it using the function `UCDisposeTextBreakLocator`.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeUtilities.h`

UCFindTextBreak

Uses locale-specific text-break information to find boundaries in Unicode text.

```
OSStatus UCFindTextBreak (
   TextBreakLocatorRef breakRef,
   UCTextBreakType breakType,
   UCTextBreakOptions options,
   const UniChar *textPtr,
   UniCharCount textLength,
   UniCharArrayOffset startOffset,
   UniCharArrayOffset *breakOffset
);
```

Parameters

breakRef
A valid reference to a text-break locator object. If the type of boundary specified by the `breakType` parameter is `BreakChar`, you can pass `NULL`. You use the function `UCCreateTextBreakLocator` (page 2156) to obtain a text-break locator object reference. If non-`NULL`, the text-break locator object must support the type of boundary specified in the `breakType` parameter.

breakType
A value of type `UCTextBreakType`, with exactly one bit set to specify a single type of boundary to be located. Since support for finding character boundaries is locale-independent and built into the `UCFindTextBreak` function, if you specify `BreakChar` as the type of boundary, then the `breakRef` parameter is ignored and may be `NULL`.

options
A `UCTextBreakOptions` value to specify the operation of the `UCFindTextBreak` function. You can use text-break locator options to control some location-independent aspects of a text-boundary search. Note that if you do not specify any `UCTextBreakOptions` values, `UCFindTextBreak` searches forward, but assumes that the `startOffset` value refers to the character preceding the offset rather than the one at the offset. This can result in `UCFindTextBreak` returning an offset that is equal to the start offset.

textPtr
A pointer to the initial character of the Unicode string to search.

textLength
The total count of Unicode characters in the string to search.

startOffset

A UniCharArrayOffset value specifying the offset from which UCFindTextBreak is to begin searching for the next text boundary of the type specified in the breakType parameter. If startOffset == 0 then kUCTextBreakLeadingEdgeMask must be set in the options parameter; if startOffset == textLength then kUCTextBreakLeadingEdgeMask must not be set.

breakOffset

A pointer to a UniCharArrayOffset value. On return, the value pointed to by the breakOffset parameter is set to the offset of the text boundary located by UCFindTextBreak. In normal usage (when exactly one of kUCTextBreakLeadingEdgeMask and kUCTextBreakGoBackwardsMask are set), the result returned in breakOffset is not equal to that supplied in the startOffset parameter unless an error occurs (and the function result is other than noErr). However, when kUCTextBreakLeadingEdgeMask and kUCTextBreakGoBackwardsMask are both set or both clear, the result produced in breakOffset can be equal to the value of startOffset.

Return Value
A result code. The text-break locator referenced by the breakRef parameter must support the type of boundary specified in the breakType parameter; otherwise, the function returns kUCTextBreakLocatorMissingType.

Discussion
The UCFindTextBreak function starts from a specified offset in a text buffer, and then proceeds forward or backward (as requested) until it finds the next text boundary of a particular locale-specific type, using a given set of options. The different types of breaks or boundaries in a line of Unicode text can include

- Boundaries of characters (treating surrogate pairs as a single character).
- Boundaries of character clusters. A cluster is a group of characters that should be treated as single text element for editing operations such as cursor movement. Typically this includes groups such as a base character followed by a sequence of combining characters, for example, a Hangul syllable represented as a sequence of conjoining jamo characters or an Indic consonant cluster.
- Boundaries of words. This can be used to determine what to highlight as the result of a double-click.
- Potential line break locations.

Finding boundaries of characters is a locale-independent operation, and support for it is built directly into the UCFindTextBreak function. If that is the only type of text boundary that you wish to locate, it is not necessary to call UCCreateTextBreakLocator and create a text-break locator object.

When finished with the text-break locator object, dispose it using the function UCDisposeTextBreakLocator (page 2158).

Availability
Available in CarbonLib 1.0 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.

Declared In
UnicodeUtilities.h

UCGetCollationKey

Uses locale-specific collation information to generate a collation key for a Unicode string.

```
OSStatus UCGetCollationKey (
   CollatorRef collatorRef,
   const UniChar *textPtr,
   UniCharCount textLength,
   ItemCount maxKeySize,
   ItemCount *actualKeySize,
   UCCollationValue collationKey[]
);
```

Parameters

collatorRef

A valid reference to a collator object; `NULL` is not allowed. You can use the function `UCCreateCollator` (page 2155) to obtain a collator reference.

textPtr

A pointer to the Unicode string (a `UniChar` array) for which to generate a collation key.

textLength

The total count of Unicode characters in the string referenced by the `textPtr` parameter.

maxKeySize

An `ItemCount` value specifying the length of the `UCCollationValue` array passed in the `collationKey` parameter. This dimension should typically be at least `5*textLength`, as the byte length of a collation key is typically more than 16 times the number of Unicode characters in the string.

actualKeySize

On return, the actual length of the `UCCollationValue` array returned in the `collationKey` parameter.

collationKey

An array of `UCCollationValue` values. On return, the array contains the new collation key. The collation key consists of a sequence of primary weights for all of the collation text elements in the string, followed by a separator and a sequence of secondary weights for all of the text elements in the string, and so on for several levels of significance. The separator is usually 0; however, 1 is used as the separator at the boundary between levels that are significant and levels that are insignificant for the options you supply in the collator object.

Return Value

A result code. The function can return `paramErr`, for example, if the parameters `collatorRef`, `textPtr`, `actualKeySize`, or `collationKey` are `NULL`. It can also return memory errors. If `maxKeySize` is too small for the collation key, the function returns `kUCOutputBufferTooSmall`.

Discussion

If you want to compare the same strings several times, as when sorting a list of strings, it may be most efficient for you to derive a collation key for each string and then compare the collation keys. A collation key is a transformation of the string that depends on the collator object (that is, it depends on the locale, the collation variant if any, and the collation options).

Collation keys that are generated using the same collator object—but for different strings—can quickly be compared with each other, without further reference to the collator object or collation tables. The disadvantage is that the collation keys may be rather large. After you use the `UCGetCollationKey` function to create a collation key from a given string and collator object, you can call the function `UCCompareCollationKeys` (page 2150) to compare two collation keys that were generated with the same collator object.

If you are comparing different strings, it may be more efficient for you to call the function `UCCompareText` (page 2151) multiple times using the same collator object.

Note that collation keys should be used only in a runtime context. They should not be stored in a persistent state (such as to disk) because the format of a collation key could change in the future.

Special Considerations

This function can move memory.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.6 or later.
Available in Mac OS X 10.0 and later.

Declared In

`UnicodeUtilities.h`

UCKeyTranslate

Converts a combination of a virtual key code, a modifier key state, and a dead-key state into a string of one or more Unicode characters.

```
OSStatus UCKeyTranslate (
   const UCKeyboardLayout *keyLayoutPtr,
   UInt16 virtualKeyCode,
   UInt16 keyAction,
   UInt32 modifierKeyState,
   UInt32 keyboardType,
   OptionBits keyTranslateOptions,
   UInt32 *deadKeyState,
   UniCharCount maxStringLength,
   UniCharCount *actualStringLength,
   UniChar unicodeString[]
);
```

Parameters

keyLayoutPtr

A pointer to the first element in a resource of type `'uchr'`. Pass a pointer to the `'uchr'` resource that you wish the `UCKeyTranslate` function to use when converting the virtual key code to a Unicode character. The resource handle associated with this pointer need not be locked, since the `UCKeyTranslate` function does not move memory.

virtualKeyCode

An unsigned 16-bit integer. Pass a value specifying the virtual key code that is to be translated. For ADB keyboards, virtual key codes are in the range from 0 to 127.

keyAction

An unsigned 16-bit integer. Pass a value specifying the current key action. See "Key Actions" (page 2178) for descriptions of possible values.

modifierKeyState

An unsigned 32-bit integer. Pass a bit mask indicating the current state of various modifier keys. You can obtain this value from the modifiers field of the event record as follows:

```
modifierKeyState = ((EventRecord.modifiers) >> 8) & 0xFF;
```

keyboardType

An unsigned 32-bit integer. Pass a value specifying the physical keyboard type (that is, the keyboard shape shown by Key Caps). You can call the function `LMGetKbdType` for this value.

keyTranslateOptions

A bit mask of options for controlling the `UCKeyTranslate` function. See "Key Translation Options Flag" (page 2181) and "Key Translation Options Mask" (page 2182) for descriptions of possible values.

deadKeyState

A pointer to an unsigned 32-bit value, initialized to zero. The `UCKeyTranslate` function uses this value to store private information about the current dead key state.

maxStringLength

A value of type `UniCharCount`. Pass the number of 16-bit Unicode characters that are contained in the buffer passed in the `unicodeString` parameter. This may be a value of up to 255, although it would be rare to get more than 4 characters.

actualStringLength

A pointer to a value of type `UniCharCount`. On return this value contains the actual number of Unicode characters placed into the buffer passed in the `unicodeString` parameter.

unicodeString

An array of values of type `UniChar`. Pass a pointer to the buffer whose sized is specified in the `maxStringLength` parameter. On return, the buffer contains a string of Unicode characters resulting from the virtual key code being handled. The number of characters in this string is less than or equal to the value specified in the `maxStringLength` parameter.

Return Value

A result code. If you pass `NULL` in the `keyLayoutPtr` parameter, `UCKeyTranslate` returns `paramErr`. The `UCKeyTranslate` function also returns `paramErr` for an invalid `'uchr'` resource format or for invalid `virtualKeyCode` or `keyAction` values, as well as for `NULL` pointers to output values.The result `kUCOutputBufferTooSmall` (-25340) is returned for an output string length greater than `maxStringLength`.

Discussion

The `UCKeyTranslate` function uses the data in a Unicode keyboard-layout (`'uchr'`) resource to map a combination of virtual key code and modifier key state to a sequence of up to 255 Unicode characters. This mapping process depends on, and may update, a dead key state; the `UCKeyTranslate` function and the `'uchr'` resource support multiple dead keys. The mapping may also depend on the specific type of key action and the type of physical keyboard being used. The `UCKeyTranslate` function supports non-ADB keyboards, an extensible set of modifier keys, and other possible extensions.

In most cases, your application does not need to call the `UCKeyTranslate` function, since the Text Services Manager automatically calls it on your behalf to handle input from a Unicode keyboard layout. However, there may be some circumstances in which your application should call `UCKeyTranslate`. For example, your application may need to determine what character(s) would have been generated for the virtual key code in the current key-down event if a different modifier-and-key combination had been used.

The basic process by which `UCKeyTranslate` uses the `'uchr'` resource to translate virtual key codes into Unicode characters is detailed in the following steps:

1. The bit pattern specifying the modifier key state is mapped by the `UCKeyModifiersToTableNum` structure to a table number.

2. The table number maps to an offset within a `UCKeyToCharTableIndex` structure that refers to the actual key-code-to-character tables.

3. The key-code-to-character tables map the virtual key code to `UCKeyOutput` values, for which there are two possibilities:

- If bits 15 and 14 of the `UCKeyOutput` value are 01, the `UCKeyOutput` value is an index into the offsets contained in a `UCKeyStateRecordsIndex` structure. If this occurs, the mapping process for the virtual key code continues on to the next step
- Otherwise, the `UCKeyOutput` value produces one or more Unicode characters, either directly or via reference to a `UCKeySequenceDataIndex` structure. This ends the mapping process for a given virtual key code.

4. The offsets in a `UCKeyStateRecordsIndex` structure refer to `UCKeyStateRecord` dead-key state records.

5. The dead-key state records map from the current dead-key state to one or more Unicode characters to be output or the following dead-key state (if any). The mapping process for a given virtual key code may end with the dead-key state record or, if there is no dead-key state record entry for the key code, with a default state terminator, as specified in the resource's `UCKeyStateTerminators` table.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.5 or later.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeUtilities.h`

Data Types

CollatorRef

Refers to an opaque object that encapsulates locale and collation information for the purpose of performing Unicode string comparison.

```
typedef struct OpaqueCollatorRef * CollatorRef;
```

Discussion
You can obtain a `CollatorRef` value from the function `UCCreateCollator` (page 2155).

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

Declared In
`UnicodeUtilities.h`

TextBreakLocatorRef

Refers to an opaque object that encapsulates locale and text-break information for the purpose of finding boundaries in Unicode text.

```
typedef struct OpaqueTextBreakLocatorRef * TextBreakLocatorRef;
```

Discussion
You can obtain a `TextBreakLocatorRef` value from the function `UCCreateTextBreakLocator` (page 2156).

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

Declared In
`UnicodeUtilities.h`

UCCollationValue

Specifies a Unicode collation key.

```
typedef UInt32 UCCollationValue;
```

Discussion
Collation keys consist of an array of `UCCollationValue` values. The collation key consists of a sequence of primary weights for all of the collation text elements in the string, followed by a separator and a sequence of secondary weights for all of the text elements in the string, and so on for several levels of significance. The separator is usually 0; however, 1 is used as the separator at the boundary between levels that are significant and levels that are insignificant for the options you supply in the collator object. You can obtain a collation key with the function `UCGetCollationKey` (page 2160).

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

Declared In
`UnicodeUtilities.h`

UCKeyboardLayout

Provides header data for a `'uchr'` resource.

```
struct UCKeyboardLayout {
    UInt16 keyLayoutHeaderFormat;
    UInt16 keyLayoutDataVersion;
    ByteOffset keyLayoutFeatureInfoOffset;
    ItemCount keyboardTypeCount;
    UCKeyboardTypeHeader keyboardTypeList[1];
};
typedef struct UCKeyboardLayout UCKeyboardLayout;
```

Fields

`keyLayoutHeaderFormat`
An unsigned 16-bit integer identifying the format of the structure. Set to `kUCLayoutHeaderFormat`.

`keyLayoutDataVersion`
An unsigned 16-bit integer identifying the version of the data in the resource, in binary code decimal format. For example, 0x0100 would equal version 1.0.

`keyLayoutFeatureInfoOffset`
: An unsigned 32-bit integer providing an offset to a structure of type `UCKeyLayoutFeatureInfo` (page 2168), if such is used in the resource. May be 0 if no `UCKeyLayoutFeatureInfo` table is included in the resource.

`keyboardTypeCount`
: An unsigned 32-bit integer specifying the number of `UCKeyboardTypeHeader` structures in the `keyboardTypeList[]` field's array.

`keyboardTypeList`
: A variable-length array containing structures of type `UCKeyboardTypeHeader`. Each `UCKeyboardTypeHeader` entry specifies a range of physical keyboard types and contains offsets to each of the key mapping sections to be used for that range of keyboard types.

Discussion
The Unicode keyboard-layout (`'uchr'`) resource contains the data necessary to map virtual key codes to Unicode character codes for a given keyboard layout. The `'uchr'` format consists of a header information section and five key mapping data sections. The `UCKeyboardLayout` type is used in the `'uchr'` resource header. It specifies version and format information, offsets to the various subtables, and an array of `UCKeyboardTypeHeader` entries.

You should use low-ASCII (0 - 0x7F) only for the KCHR/uchr resource names and you should use Unicode in the Info.plist file when you specify strings for the user-interface (UI).

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

Declared In
`UnicodeUtilities.h`

UCKeyboardTypeHeader

Specifies a range of physical keyboard types in a `'uchr'` resource.

```
struct UCKeyboardTypeHeader {
    UInt32 keyboardTypeFirst;
    UInt32 keyboardTypeLast;
    ByteOffset keyModifiersToTableNumOffset;
    ByteOffset keyToCharTableIndexOffset;
    ByteOffset keyStateRecordsIndexOffset;
    ByteOffset keyStateTerminatorsOffset;
    ByteOffset keySequenceDataIndexOffset;
};
typedef struct UCKeyboardTypeHeader UCKeyboardTypeHeader;
```

Fields
`keyboardTypeFirst`
: An unsigned 32-bit integer specifying the first keyboard type in this entry. For the initial entry (that is, the default entry) in an array of `UCKeyboardTypeHeader` structures, you should set this value to 0. The initial `UCKeyboardTypeHeader` entry is used if the keyboard type passed to the function `UCKeyTranslate` (page 2162) does not match any other entry, that is, if it is not within the range of values specified by `keyboardTypeFirst` and `keyboardTypeLast` for any entry.

`keyboardTypeLast`
: An unsigned 32-bit integer specifying the last keyboard type in this entry. For the initial entry (that is, the default entry) in an array of `UCKeyboardTypeHeader` structures, you should set this value to 0.

`keyModifiersToTableNumOffset`
: An unsigned 32-bit integer providing an offset to a structure of type `UCKeyModifiersToTableNum` (page 2169). The `'uchr'` resource requires a `UCKeyModifiersToTableNum` structure, therefore this field must contain a non-zero value.

`keyToCharTableIndexOffset`
: An unsigned 32-bit integer providing an offset to a structure of type `UCKeyToCharTableIndex` (page 2176). The `'uchr'` resource requires a `UCKeyToCharTableIndex` structure, therefore this field must contain a non-zero value.

`keyStateRecordsIndexOffset`
: An unsigned 32-bit integer providing an offset to a structure of type `UCKeyStateRecordsIndex` (page 2174), if such is used in the resource. This value may be 0 if no dead-key state records are included in the resource.

`keyStateTerminatorsOffset`
: An unsigned 32-bit integer providing an offset to a structure of type `UCKeyStateTerminators` (page 2175), if such is used in the resource. This value may be 0 if no dead-key state terminators are included in the resource.

`keySequenceDataIndexOffset`
: An unsigned 32-bit integer providing an offset to a structure of type `UCKeySequenceDataIndex` (page 2170), if such is used in the resource. This value may be 0 if no character key sequences are included in the resource.

Discussion
The `UCKeyboardTypeHeader` type is used in a structure of type `UCKeyboardLayout` (page 2165) to specify a range of physical keyboard types and contains offsets to each of the key mapping sections to be used for that range of keyboard types. Typically, you use an array of `UCKeyboardTypeHeader` structures, of which the first entry in the array is the default and will be used if the keyboard type does not fall within the range for any other entry. See `UCKeyboardLayout` (page 2165) for a further discussion of the context for use of the `UCKeyboardTypeHeader` type.

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

Declared In
`UnicodeUtilities.h`

UCKeyCharSeq

Specifies the output of a dead-key state in a `'uchr'` resource.

```
typedef UInt16 UCKeyCharSeq;
```

Discussion
The Unicode keyboard-layout (`'uchr'`) resource contains the data necessary to map virtual key codes to Unicode character codes for a given keyboard layout. The `'uchr'` format consists of a header information section and five key mapping data sections. The `UCKeyCharSeq` type is a 16-bit value used in the third key mapping section of the `'uchr'` resource to specify the output of a dead-key state.

Specifically, the dead-key state record—a structure of type `UCKeyStateRecord` (page 2173) —uses a `UCKeyCharSeq` value to contain the character output that results from the resolution of a given dead-key state. You can use a `UCKeyCharSeq` value in a dead-key state record to represent either an index to a Unicode character sequence or a single Unicode character. The `UCKeyCharSeq` type is similar to the type `UCKeyOutput` (page 2169) , but does not itself support indices into dead-key state records.

The interpretation of `UCKeyCharSeq` depends on bits 15 and 14.

If they are 10 (that is, for values in the range of 0x8000–0xBFFF), then bits 0–13 are an index into the `charSequenceOffsets[` field of a structure of type `UCKeySequenceDataIndex` (page 2170) , which contains offsets to a separate resource-wide list of Unicode character sequences. If a `UCKeySequenceDataIndex` structure is not present in the resource or the index is beyond the end of the list, then the entire value (that is, bits 0–15) is a single Unicode character to emit. Otherwise (for values in the range of 0x0000–0x7FFF and 0xC000–0xFFFD), bits 0–15 are a single Unicode character, with the exception that a value of 0xFFFE–0xFFFF means no character output (these are invalid Unicode codes).

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

Declared In
`UnicodeUtilities.h`

UCKeyLayoutFeatureInfo

Specifies the longest possible output string to be produced by the current `'uchr'` resource.

```
struct UCKeyLayoutFeatureInfo {
    UInt16 keyLayoutFeatureInfoFormat;
    UInt16 reserved;
    UniCharCount maxOutputStringLength;
};
typedef struct UCKeyLayoutFeatureInfo UCKeyLayoutFeatureInfo;
```

Fields

`keyLayoutFeatureInfoFormat`
: An unsigned 16-bit integer identifying the format of the `UCKeyLayoutFeatureInfo` structure. Set to `kUCKeyLayoutFeatureInfoFormat`.

`reserved`
: Reserved. Set to 0.

`maxOutputStringLength`
: An unsigned 32-bit integer specifying the longest possible output string of Unicode characters to be produced by this `'uchr'` resource.

Discussion
The Unicode keyboard-layout (`'uchr'`) resource contains the data necessary to map virtual key codes to Unicode character codes for a given keyboard layout. The `'uchr'` format consists of a header information section and five key mapping data sections. The `UCKeyLayoutFeatureInfo` type is used in the header section of the `'uchr'` resource.

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

Declared In
`UnicodeUtilities.h`

UCKeyModifiersToTableNum

Maps a modifier key combination to a particular key-code-to-character table number in a 'uchr' resource.

```
struct UCKeyModifiersToTableNum {
    UInt16 keyModifiersToTableNumFormat;
    UInt16 defaultTableNum;
    ItemCount modifiersCount;
    UInt8 tableNum[1];
};
typedef struct UCKeyModifiersToTableNum UCKeyModifiersToTableNum;
```

Fields

keyModifiersToTableNumFormat
: An unsigned 16-bit integer identifying the format of the UCKeyModifiersToTableNum structure. Set to kUCKeyModifiersToTableNumFormat.

defaultTableNum
: An unsigned 16-bit integer identifying the table number to use for modifier combinations that are outside of the range included in the tableNum field.

modifiersCount
: An unsigned 32-bit integer specifying the range of modifier bit combinations for which there are entries in the tableNum[] field.

tableNum
: An array of unsigned 8-bit integers that map modifier bit combinations to table numbers. These values are indexes into the keyToCharTableOffsets array in a UCKeyToCharTableIndex (page 2176)structure; these, in turn, are offsets to the actual key-code-to character tables, which follow the UCKeyToCharTableIndex structure in the 'uchr' resource.

Discussion

The Unicode keyboard-layout ('uchr') resource contains the data necessary to map virtual key codes to Unicode character codes for a given keyboard layout. The 'uchr' format consists of a header information section and five key mapping data sections. The UCKeyModifiersToTableNum type is used in the first key mapping section of the 'uchr' resource. It maps a modifier key combination to a particular key-code-to-character table number.

Availability

Available in Mac OS X v10.0 and later.

Declared In

UnicodeUtilities.h

UCKeyOutput

Specifies values in key-code-to-character tables in a 'uchr' resource.

```
typedef UInt16 UCKeyOutput;
```

Discussion

The Unicode keyboard-layout ('uchr') resource contains the data necessary to map virtual key codes to Unicode character codes for a given keyboard layout. The 'uchr' format consists of a header information section and five key mapping data sections. The UCKeyOutput type is a 16-bit value used in the second key mapping section of a 'uchr' resource to specify values in key-code-to-character tables.

You use a `UCKeyOutput` value in a key-code-to-character table to represent one of the following: an index to a dead-key state record, an index to a Unicode character sequence, or a single Unicode character.

The interpretation of a `UCKeyOutput` value depends on bits 15 and 14.

If they are 01 (that is, for values in the range of 0x4000-0x7FFF), then bits 0-13 are an index into the `keyStateRecordOffsets` field of a structure of type `UCKeyStateRecordsIndex` (page 2174) , which contains offsets to a separate resource-wide list of dead-key state records.

If they are 10 (that is, for values in the range of 0x8000-0xBFFF), then bits 0-13 are an index into the `charSequenceOffsets` field of a structure of type `UCKeySequenceDataIndex` (page 2170) , which contains offsets to a separate resource-wide list of Unicode character sequences. If a `UCKeySequenceDataIndex` structure is not present in the resource or the index is beyond the end of the list, then the entire value (that is, bits 0-15) is a single Unicode character to emit.

Otherwise (for values in the range of 0x0000-0x3FFF and 0xC000-0xFFFD), bits 0-15 are a single Unicode character, with the exception that a value of 0xFFFE-0xFFFF means no character output (these are invalid Unicode codes).

Most single Unicode characters that are likely to be generated by direct keyboard input are in the range 0x0000-0x33FF or 0xE000-0xFFFD, and so are covered by the single-character cases above. Characters outside this range can still be generated by direct keyboard input, in which case they must be represented as 1-character sequences. The fifth key mapping section of the `'uchr'` resource, introduced by the `UCKeySequenceDataIndex` type, provides for this option.

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

Declared In
`UnicodeUtilities.h`

UCKeySequenceDataIndex

Contains offsets to a list of character sequences for a `'uchr'` resource.

```
struct UCKeySequenceDataIndex {
    UInt16 keySequenceDataIndexFormat;
    UInt16 charSequenceCount;
    UInt16 charSequenceOffsets[1];
};
typedef struct UCKeySequenceDataIndex UCKeySequenceDataIndex;
```

Fields

`keySequenceDataIndexFormat`
An unsigned 16-bit integer identifying the format of the `UCKeySequenceDataIndex` structure. Set to `kUCKeySequenceDataIndexFormat`.

`charSequenceCount`
An unsigned 16-bit integer specifying the number of Unicode character sequences that follow the end of the `UCKeySequenceDataIndex` structure.

`charSequenceOffsets`
: An array of offsets from the beginning of the `UCKeySequenceDataIndex` structure to the Unicode character sequences that follow it. Because a given offset indicates both the beginning of a new character sequence and the end of the sequence that precedes it, the length of each sequence is determined by the difference between the offset to that sequence and the value of the next offset in the array. The array contains one more entry than the number of character sequences; the final entry is the offset to the end of the final character sequence.

Discussion
The Unicode keyboard-layout (`'uchr'`) resource contains the data necessary to map virtual key codes to Unicode character codes for a given keyboard layout. The `'uchr'` format consists of a header information section and five key mapping data sections. The `UCKeySequenceDataIndex` type is used in the fifth key mapping section of the `'uchr'` resource.

The `UCKeySequenceDataIndex` structure contains offsets to a list of character sequences for the `'uchr'` resource. This permits a single keypress to generate a sequence of characters, or to generate a single character outside the range that can be represented directly by values of type `UCKeyOutput` (page 2169) or `UCKeyCharSeq` (page 2167).

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

Declared In
`UnicodeUtilities.h`

UCKeyStateEntryRange

Maps from a dead-key state to either the resultant Unicode character(s) or the new dead key state produced when the current state is terminated by a given character key for a `'uchr'` resource.

```
struct UCKeyStateEntryRange {
    UInt16 curStateStart;
    UInt8 curStateRange;
    UInt8 deltaMultiplier;
    UCKeyCharSeq charData;
    UInt16 nextState;
};
typedef struct UCKeyStateEntryRange UCKeyStateEntryRange;
```

Fields

`curStateStart`
: An unsigned 16-bit integer specifying the beginning of a given dead-key state range.

`curStateRange`
: An unsigned 8-bit integer specifying the number of entries in a given dead-key state range.

`deltaMultiplier`
: An unsigned 8-bit integer.

`charData`
: A value of type `UCKeyCharSeq`. This base character value is used to determine the actual Unicode character(s) produced when a given dead-key state terminates.

`nextState`
: An unsigned 16-bit integer. This base dead-key state value is used to determine the following dead-key state, if any.

Discussion
The `UCKeyStateEntryRange` type is used in the `stateEntryData[]` field of a structure of type `UCKeyStateRecord` (page 2173). You should use the `UCKeyStateEntryRange` format for complex (multiple) dead-key states.

For each virtual key code, an entry in its dead-key state record maps from the current dead-key state to the Unicode character(s) produced or to the next dead-key state, as follows.

If the current dead-key state is within a valid dead-key state range for the given input character—that is, if its value is greater than or equal to `curStateStart` and less than or equal to `curStateStart + curStateRange`—then

- If the base `charData` value for the given dead-key state range is in the range of valid Unicode characters, a character is produced and the dead-key state may be terminated.

and/or

- If the base `nextState` value is not 0, a new dead-key state is produced.

In the first case, the output character is determined as follows: The base `charData` value is incremented by the resulting product of (the difference between the current state and the start of that state's range) and (a multiplier). That is:

```
charData += (curState - curStateStart) * deltaMultiplier
```

Similarly, in the second case, the resulting dead-key state, which is the new `curState` value, is determined as follows: The base `nextState` value is incremented by the resulting product of (the difference between the current state and the start of that state's range) and (a multiplier). That is:

```
nextState += (curState - curStateStart) * deltaMultiplier
```

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

Declared In
`UnicodeUtilities.h`

UCKeyStateEntryTerminal

Maps from a dead-key state to the Unicode character(s) produced when that state is terminated by a given character key for a `'uchr'` resource.

```
struct UCKeyStateEntryTerminal {
    UInt16 curState;
    UCKeyCharSeq charData;
};
typedef struct UCKeyStateEntryTerminal UCKeyStateEntryTerminal;
```

Fields
`curState`
An unsigned 16-bit integer specifying the current dead-key state.

`charData`

A value of type `UCKeyCharSeq` specifying the Unicode character(s) produced when a given character key is pressed.

Discussion

The `UCKeyStateEntryTerminal` type is used in the `stateEntryData[]` field of a structure of type `UCKeyStateRecord` (page 2173). You should use the `UCKeyStateEntryTerminal` format for simple dead-key states that are terminated by a single keystroke, as in the U.S. keyboard layout. Each entry maps from the current dead-key state to the Unicode character(s) produced when a given character key is pressed that terminates the dead-key state.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`UnicodeUtilities.h`

UCKeyStateRecord

Determines dead-key state transitions in a `'uchr'` resource.

```
struct UCKeyStateRecord {
    UCKeyCharSeq stateZeroCharData;
    UInt16 stateZeroNextState;
    UInt16 stateEntryCount;
    UInt16 stateEntryFormat;
    UInt32 stateEntryData[1];
};
typedef struct UCKeyStateRecord UCKeyStateRecord;
```

Fields

`stateZeroCharData`

A value of type `UCKeyCharSeq` specifying the Unicode character(s) produced from a given key code while no dead-key state is in effect.

`stateZeroNextState`

An unsigned 16-bit integer specifying the dead-key state produced from a given key code when no previous dead-key state is in effect. If the `UCKeyStateRecord` structure does not initiate a dead-key state (but only provides terminators for other dead-key states), this will be 0. A non-zero value specifies the resulting new dead-key state and refers to the current state entry within the `stateEntryData[]` field for the following dead-key state record that is applied.

`stateEntryCount`

An unsigned 16-bit integer specifying the number of elements in the `stateEntryData` field's array for a given dead-key state record.

`stateEntryFormat`

An unsigned 16-bit integer specifying the format of the data in the `stateEntryData` field's array. This should be 0 if the `stateEntryCount` field is set to 0. Currently available values are `kUCKeyStateEntryTerminalFormat` and `kUCKeyStateEntryRangeFormat`; see "Key State Entry Formats" (page 2181) for descriptions of these values.

`stateEntryData`

An array of dead-key state entries, whose size depends on their format, but which will always be a multiple of 4 bytes. Each entry maps from the current dead-key state to the Unicode character(s) that result when a given character key is pressed or to the next dead-key state, if any. The format of the entry is specified by the `stateEntryFormat` field to be either that of type `UCKeyStateEntryTerminal` (page 2172) or `UCKeyStateEntryRange` (page 2171).

Discussion

The Unicode keyboard-layout (`'uchr'`) resource contains the data necessary to map virtual key codes to Unicode character codes for a given keyboard layout. The `'uchr'` format consists of a header information section and five key mapping data sections. The `UCKeyStateRecord` type is used in the third key mapping section of the `'uchr'` resource to determine dead-key state transitions. The `UCKeyStateRecord` structure permits complex dead-key state processing, such as a series of transitions from one dead-key state directly into another, in which each transition can emit a sequence of one or more Unicode characters.

Any modifier key combination which initiates a dead-key state or which is a valid terminator of a dead-key state refers to one of these records via the `UCKeyOutput` (page 2169) values in key-code-to-character tables. A `UCKeyOutput` value may index the offsets contained in a `UCKeyStateRecordsIndex` (page 2174) structure, which in turn refers to the actual dead-key state records.

Each `UCKeyStateRecord` structure maps from the current dead-key state to the character data to be output or the following dead-key state (if any), as follows:

- If the current dead-key state is zero (that is, there are no dead keys in effect) the value in `stateZeroCharData` is output and the state is set to the value in `stateZeroNextState` (this can be used to initiate a dead-key state).
- If the current dead-key state is non-zero and there is an entry for the state in `stateEntryData`, then the corresponding value in `stateEntryData.charData` is output. The next state is then set to either a `kUCKeyStateEntryTerminalFormat` or a `kUCKeyStateEntryRangeFormat` value; in either case, if the next dead-key state is 0, this implements a valid dead-key state terminator.
- If the current dead-key state is non-zero, and there is no entry for the state in `stateEntryData`, the default state terminator is output from the `'uchr'` resource's `UCKeyStateTerminators` (page 2175) table for the current state (or nothing may be output, if there is no `UCKeyStateTerminators` table or it has no entry for the current state). Then the value in `stateZeroCharData` is output, and the state is set to the value in `stateZeroNextState`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`UnicodeUtilities.h`

UCKeyStateRecordsIndex

Provides a count of, and offsets to, dead-key state records in a `'uchr'` resource.

```
struct UCKeyStateRecordsIndex {
    UInt16 keyStateRecordsIndexFormat;
    UInt16 keyStateRecordCount;
    ByteOffset keyStateRecordOffsets[1];
};
typedef struct UCKeyStateRecordsIndex UCKeyStateRecordsIndex;
```

Fields

`keyStateRecordsIndexFormat`

An unsigned 16-bit integer identifying the format of the `UCKeyStateRecordsIndex` structure. Set to `kUCKeyStateRecordsIndexFormat`.

`keyStateRecordCount`

An unsigned 16-bit integer specifying the number of dead-key state records that are included in the resource.

`keyStateRecordOffsets`

An array of offsets from the beginning of the resource to each of the `UCKeyStateRecord` values that follow this structure in the `'uchr'` resource.

Discussion

The Unicode keyboard-layout (`'uchr'`) resource contains the data necessary to map virtual key codes to Unicode character codes for a given keyboard layout. The `'uchr'` format consists of a header information section and five key mapping data sections. The `UCKeyStateRecordsIndex` type is used in the third key mapping section of the `'uchr'` resource.

The `UCKeyStateRecordsIndex` structure is an index to dead-key state records of type `UCKeyStateRecord` (page 2173). Any keycode-modifier combination which initiates a dead-key state or which is a valid terminator of a dead-key state refers to one of these records, via the `UCKeyStateRecordsIndex` structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`UnicodeUtilities.h`

UCKeyStateTerminators

Lists the default terminators for each dead-key state handled by a `'uchr'` resource.

```
struct UCKeyStateTerminators {
    UInt16 keyStateTerminatorsFormat;
    UInt16 keyStateTerminatorCount;
    UCKeyCharSeq keyStateTerminators[1];
};
typedef struct UCKeyStateTerminators UCKeyStateTerminators;
```

Fields

`keyStateTerminatorsFormat`

An unsigned 16-bit integer identifying the format of the `UCKeyStateTerminators` structure. Set to `kUCKeyStateTerminatorsFormat`.

`keyStateTerminatorCount`

An unsigned 16-bit integer specifying the number of default dead-key state terminators contained in the `keyStateTerminators[]` array.

`keyStateTerminators`
: An array of default dead-key state terminators, described as values of type `UCKeyCharSeq` (page 2167); the value `keyStateTerminators[0]` is the terminator for state 1, and so on.

Discussion
The Unicode keyboard-layout (`'uchr'`) resource contains the data necessary to map virtual key codes to Unicode character codes for a given keyboard layout. The `'uchr'` format consists of a header information section and five key mapping data sections. The `UCKeyStateTerminators` type is used in the fourth key mapping section of the `'uchr '` resource.

The `UCKeyStateTerminators` structure contains the list of default terminators (characters or sequences) for each dead-key state that is handled by a `'uchr'` resource. When a dead-key state is in effect but a modifier-and-key combination is typed which has no special handling for that state, the default terminator for the state is output before the modifier-and-key combination is processed. If this table is not present or does not extend far enough to have a terminator for the state, nothing is output when the state terminates.

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

Declared In
`UnicodeUtilities.h`

UCKeyToCharTableIndex

Provides a count of, and offsets to, key-code-to-character tables in a `'uchr'` resource.

```
struct UCKeyToCharTableIndex {
    UInt16 keyToCharTableIndexFormat;
    UInt16 keyToCharTableSize;
    ItemCount keyToCharTableCount;
    ByteOffset keyToCharTableOffsets[1];
};
typedef struct UCKeyToCharTableIndex UCKeyToCharTableIndex;
```

Fields
`keyToCharTableIndexFormat`
: An unsigned 16-bit integer identifying the format of the `UCKeyToCharTableIndex` structure. Set to `kUCKeyToCharTableIndexFormat`.

`keyToCharTableSize`
: An unsigned 16-bit integer specifying the number of virtual key codes supported by this resource; for ADB keyboards this is 128 (with virtual key codes ranging from 0 to 127).

`keyToCharTableCount`
: An unsigned 32-bit integer specifying the number of key-code-to-character tables, typically 6 to 12.

`keyToCharTableOffsets`
: An array of offsets from the beginning of the `'uchr'` resource to each of the `UCKeyOutput` key-code-to-character tables in the `keyToCharData[]` array that follows this structure in the resource.

Discussion
The Unicode keyboard-layout (`'uchr'`) resource contains the data necessary to map virtual key codes to Unicode character codes for a given keyboard layout. The `'uchr'` format consists of a header information section and five key mapping data sections. The `UCKeyToCharTableIndex` type is used in the second key mapping section of the `'uchr'` resource. The `UCKeyToCharTableIndex` structure precedes the list of key-code-to-character tables, each of which maps a key code to a 16-bit value of type `UCKeyOutput` (page 2169).

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

Declared In
`UnicodeUtilities.h`

Constants

Fixed Ordering Scheme

Specifies to use the fixed ordering scheme.

```
enum {
    kUCCollateTypeHFSExtended = 1
};
```

Constants
`kUCCollateTypeHFSExtended`
The `kUCCollateTypeHFSExtended` ordering scheme sorts maximally decomposed Unicode according to the rules used by the HFS Extended volume format for its catalog. When this order is used, other collation options are ignored; this order is always case-insensitive (for decomposed characters) and ignores the Unicode characters 200C-200F, 202A-202E, 206A-206F, FEFF.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

Discussion
`UCCollateOptions` is a 32-bit value. Bits 0-23 are described in "String Comparison Options" (page 2183). The field consisting of bits 24-31 is used for values that specify which fixed ordering scheme to use with the function `UCCompareTextNoLocale` (page 2154). Currently only one such scheme is provided.

Constants are provided for setting and testing the `UCCollateOptions` field that specifies the ordering scheme. These values are described in "Fixed Ordering Masks 1" (page 2177) and "Fixed Ordering Masks 2" (page 2178).

Fixed Ordering Masks 1

Set and test the `UCCollateOptions` field that specifies a fixed ordering scheme.

```
enum {
    kUCCollateTypeSourceMask = 0x000000FF,
    kUCCollateTypeShiftBits = 24
};
```

Constants
`kUCCollateTypeSourceMask`
You can use this mask, in conjunction with the `kUCCollateTypeShiftBits` constant, to obtain a value identifying a fixed ordering scheme.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCCollateTypeShiftBits`

You can use this value, along with one of the constants described in "Fixed Ordering Scheme" (page 2177), to specify a fixed ordering scheme. You can also use this value, in conjunction with the `kUCCollateTypeSourceMask` constant, to obtain a value identifying a fixed ordering scheme.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

Discussion
You can use these constants to set or obtain a value that specifies a fixed ordering scheme. For a description of the available types of fixed ordering schemes, see "Fixed Ordering Scheme" (page 2177).

For example, to specify `kUCCollateTypeHFSExtended` in the `options` parameter of the function `UCCompareTextNoLocale` (page 2154) , the `kUCCollateTypeHFSExtended` value must be shifted by `kUCCollateTypeShiftBits` :

```
options = kUCCollateTypeHFSExtended kUCCollateTypeShiftBits;
```

You would obtain the ordering scheme value from the `options` parameter as follows:

```
fixedOrderType = ((options > > kUCCollateTypeShiftBits) &
kUCCollateTypeSourceMask);
```

See also "Fixed Ordering Masks 2" (page 2178).

Fixed Ordering Masks 2

Test the `UCCollateOptions` field that specifies a fixed ordering scheme.

```
enum {
    kUCCollateTypeMask = kUCCollateTypeSourceMask << kUCCollateTypeShiftBits
};
```

Constants
`kUCCollateTypeMask`

You can use this mask to directly test bits 24-31 of a `UCCollateOptions` value.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

Discussion
See also "Fixed Ordering Scheme" (page 2177).

See also "Fixed Ordering Masks 1" (page 2177).

Key Actions

Indicate the current key action.

```
enum {
    kUCKeyActionDown = 0,
    kUCKeyActionUp = 1,
    kUCKeyActionAutoKey = 2,
    kUCKeyActionDisplay = 3
};
```

Constants

`kUCKeyActionDown`
The user is pressing the key.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeUtilities.h`.

`kUCKeyActionUp`
The user is releasing the key.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeUtilities.h`.

`kUCKeyActionAutoKey`
The user has the key in an "auto-key" pressed state that is, the user is holding down the key for an extended period of time and is thereby generating multiple key strokes from the single key.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeUtilities.h`.

`kUCKeyActionDisplay`
The user is requesting information for key display, as in the Key Caps application.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeUtilities.h`.

Discussion

You can supply the following constants for the `keyAction` parameter of the function `UCKeyTranslate` (page 2162) to indicate the current key action.

Key Format Codes

Indicate a structure format used in a `'uchr'` resource.

```
enum {
    kUCKeyLayoutHeaderFormat = 0x1002,
    kUCKeyLayoutFeatureInfoFormat = 0x2001,
    kUCKeyModifiersToTableNumFormat = 0x3001,
    kUCKeyToCharTableIndexFormat = 0x4001,
    kUCKeyStateRecordsIndexFormat = 0x5001,
    kUCKeyStateTerminatorsFormat = 0x6001,
    kUCKeySequenceDataIndexFormat = 0x7001
};
```

Constants

`kUCKeyLayoutHeaderFormat`
The format of a structure of type `UCKeyboardLayout` (page 2165).
Available in Mac OS X v10.0 and later.
Declared in `UnicodeUtilities.h`.

`kUCKeyLayoutFeatureInfoFormat`
The format of a structure of type `UCKeyLayoutFeatureInfo` (page 2168).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCKeyModifiersToTableNumFormat`
The format of a structure of type `UCKeyModifiersToTableNum` (page 2169).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCKeyToCharTableIndexFormat`
The format of a structure of type `UCKeyToCharTableIndex` (page 2176).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCKeyStateRecordsIndexFormat`
The format of a structure of type `UCKeyStateRecordsIndex` (page 2174).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCKeyStateTerminatorsFormat`
The format of a structure of type `UCKeyStateTerminators` (page 2175).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCKeySequenceDataIndexFormat`
The format of a structure of type `UCKeySequenceDataIndex` (page 2170).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

Discussion
These constants are those currently defined to be used within the various structures in a `'uchr'` resource to indicate each structure's format.

Key Output Index Masks

Test the bits in `UCKeyOutput` values.

```
enum {
    kUCKeyOutputStateIndexMask = 0x4000,
    kUCKeyOutputSequenceIndexMask = 0x8000,
    kUCKeyOutputTestForIndexMask = 0xC000,
    kUCKeyOutputGetIndexMask = 0x3FFF
};
```

Constants
`kUCKeyOutputStateIndexMask`
If the bit specified by this mask is set, the `UCKeyStateRecordsIndex` (page 2174) `UCKeyOutput` value contains an index into a structure of type `UCKeyStateRecordsIndex` (page 2174).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCKeyOutputSequenceIndexMask`
If the bit specified by this mask is set, the `UCKeyOutput` value contains an index into a structure of type `UCKeySequenceDataIndex` (page 2170).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCKeyOutputTestForIndexMask`
You can use this mask to test the bits (14–15) in the `UCKeyOutput` value that determine whether the value contains an index to any other structure. If both bits specified by this mask are clear, the `UCKeyOutput` value does not contain an index to any other structure.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCKeyOutputGetIndexMask`
You can use this mask to test the bits (0–13) in a `UCKeyOutput` value that provide the actual index to another structure.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

Discussion
You can use these masks to test the bits in `UCKeyOutput` values.

Key State Entry Formats

Indicate the format for dead-key state records.

```
enum {
    kUCKeyStateEntryTerminalFormat = 0x0001,
    kUCKeyStateEntryRangeFormat = 0x0002
};
```

Constants
`kUCKeyStateEntryTerminalFormat`
Specifies that the entry format is that of a structure of type `UCKeyStateEntryTerminal` (page 2172). Use this format for simple (single) dead-key states, as in the U.S. keyboard layout.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCKeyStateEntryRangeFormat`
Specifies that the entry format is that of a structure of type `UCKeyStateEntryRange` (page 2171). Use this format for complex (multiple) dead-key states, as in the hex input and Hangul input keyboard layouts.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

Discussion
These constants are used in `UCKeyStateRecord` structures to indicate the format for dead-key state records.

Key Translation Options Flag

Indicates the dead-key processing state.

```
enum {
    kUCKeyTranslateNoDeadKeysBit = 0
};
```

Constants

`kUCKeyTranslateNoDeadKeysBit`

The bit number of the bit that turns off dead-key processing. This prevents setting any new dead-key states, but allows completion of any dead-key states currently in effect.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

Discussion

Theis constant is the currently defined bit assignment for the `keyTranslateOptions` parameter of the function `UCKeyTranslate` (page 2162).

Key Translation Options Mask

Specifies the mask for the bit that controls dead-key processing state.

```
enum {
    kUCKeyTranslateNoDeadKeysMask = 1L << kUCKeyTranslateNoDeadKeysBit
};
```

Constants

`kUCKeyTranslateNoDeadKeysMask`

The mask for the bit that turns off dead-key processing. This prevents setting any new dead-key states, but allows completion of any dead-key states currently in effect.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

Discussion

This constant is the currently defined mask for the `keyTranslateOptions` parameter of the function `UCKeyTranslate` (page 2162).

Operation Class

Identifies collation as a class of Unicode utility operations.

```
enum {
    kUnicodeCollationClass = 'ucol'
};
```

Constants

`kUnicodeCollationClass`

Identifies collation as a class of operations.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

Discussion

The locales and collation variants available for collation operations can be determined by calling the Locales Utilities functions `LocaleOperationCountLocales` and `LocaleOperationGetLocales` with the `opClass` parameter set to the `kUnicodeCollationClass` constant.

Standard Options Mask

Specifies standard options for Unicode string comparison.

```
enum {
    kUCCollateStandardOptions = kUCCollateComposeInsensitiveMask
| kUCCollateWidthInsensitiveMask
};
```

Constants

`kUCCollateStandardOptions`

If the `kUCCollateComposeInsensitiveMask` and `kUCCollateWidthInsensitiveMask` bits are set, then (1) precomposed and decomposed representations of the same text element will be treated as equivalent, and (2) fullwidth and halfwidth compatibility forms will be treated as equivalent to the corresponding non-compatibility characters.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

Discussion

For descriptions of other collation options, see "String Comparison Options" (page 2183).

String Comparison Options

Specifies options for Unicode string comparison.

```
typedef UInt32 UCCollateOptions;
enum {
    kUCCollateComposeInsensitiveMask = 1L << 1,
    kUCCollateWidthInsensitiveMask = 1L << 2,
    kUCCollateCaseInsensitiveMask = 1L << 3,
    kUCCollateDiacritInsensitiveMask = 1L << 4,
    kUCCollatePunctuationSignificantMask = 1L << 15,
    kUCCollateDigitsOverrideMask = 1L << 16,
    kUCCollateDigitsAsNumberMask = 1L << 17
};
```

Constants

`kUCCollateComposeInsensitiveMask`

If the corresponding bit is set, then precomposed and decomposed representations of the same text element are treated as equivalent. This option is among those set by the `kUCCollateStandardOptions` constant, as described in "Standard Options Mask" (page 2183).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCCollateWidthInsensitiveMask`

If the corresponding bit is set, then fullwidth and halfwidth compatibility forms are treated as equivalent to the corresponding non-compatibility characters. This option is among those set by the `kUCCollateStandardOptions` constant, as described in "Standard Options Mask" (page 2183).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCCollateCaseInsensitiveMask`
If the corresponding bit is set, then uppercase and titlecase characters are treated as equivalent to the corresponding lowercase characters.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCCollateDiacritInsensitiveMask`
If the corresponding bit is set, then characters with diacritics are treated as equivalent to the corresponding characters without diacritics.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCCollatePunctuationSignificantMask`
If the corresponding bit is set, then punctuation and symbols are treated as significant instead of ignorable. This will produce results closer to the behavior of the older non-Unicode Mac OS collation functions. This option is available with Mac OS 9 and later.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCCollateDigitsOverrideMask`
If the corresponding bit is set, then the number-handling behavior is specified by the remaining number-handling option bits, instead of by the collation information for the locale. If the bit is clear, the locale controls how numbers are handled and the remaining number-handling option bits are ignored. This option is available with Mac OS 9 and later.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCCollateDigitsAsNumberMask`
If the corresponding bit is set (and if the bit corresponding to `kUCCollateDigitsOverrideMask` is also set), then numeric substrings up to six digits long are collated by their numeric value—that is, they are treated as a single text element whose primary weight depends on the numeric value of the digit string. This primary weight will be greater than the weight of any valid Unicode character, but less than the primary weight of any unassigned Unicode character. For example, this will result in "Chapter 9" sorting before "Chapter 10." Currently, these digit strings can include digits with numeric value 0-9 in any script (excluding the ideographic characters for 1-9). If the bit is clear, digits are treated like other characters for sorting. Numeric substrings longer than 6 digits are always treated as normal characters. This option is available with Mac OS 9 and later.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

Discussion
For a description of the `UCCollateOptions` values, see "Standard Options Mask" (page 2183).

Text Break Options

Specifies options for locating boundaries in Unicode text.

```
typedef UInt32 UCTextBreakOptions;
enum {
    kUCTextBreakLeadingEdgeMask = 1L << 0,
    kUCTextBreakGoBackwardsMask = 1L << 1,
    kUCTextBreakIterateMask = 1L << 2
};
```

Constants

`kUCTextBreakLeadingEdgeMask`

If the corresponding bit is set, then the starting offset for the `UCFindTextBreak` function is assumed to be in the word containing the character following the offset; this is the normal case when searching forward. If the corresponding bit is clear, then the starting offset for `UCFindTextBreak` is assumed to be in the word containing the character preceding the offset; this is the normal case when searching backward.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCTextBreakGoBackwardsMask`

If the corresponding bit is set, then `UCFindTextBreak` searches backward from the value provided in its `startOffset` parameter to find the next text break. If the corresponding bit is clear, then `UCFindTextBreak` searches forward from the `startOffset` value to find the next text break.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCTextBreakIterateMask`

The corresponding bit may be set to indicate to the `UCFindTextBreak` function that the specified starting offset is a known break of the type specified in the `breakType` parameter. This permits `UCFindTextBreak` to optimize its search for the subsequent break of the same type. When iterating through all the breaks of a particular type in a particular buffer, this bit should be set for all calls except the first (since the initial `startOffset` value may not be a known break of the specified type).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

Text Break Types

Specifies kinds of text boundaries.

```
typedef UInt32 UCTextBreakType;
enum {
    kUCTextBreakCharMask = 1L << 0,
    kUCTextBreakClusterMask = 1L << 2,
    kUCTextBreakWordMask = 1L << 4,
    kUCTextBreakLineMask = 1L << 6
};
```

Constants

`kUCTextBreakCharMask`

If the bit specified by this mask is set, boundaries of characters may be located (with surrogate pairs treated as a single character).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCTextBreakClusterMask`
: If the bit specified by this mask is set, boundaries of character clusters may be located. A cluster is a group of characters that should be treated as single text element for editing operations such as cursor movement. Typically this includes groups such as a base character followed by a sequence of combining characters, for example, a Hangul syllable represented as a sequence of conjoining jamo characters or an Indic consonant cluster.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCTextBreakWordMask`
: If the bit specified by this mask is set, boundaries of words may be located. This can be used to determine what to highlight as the result of a double-click.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

`kUCTextBreakLineMask`
: If the bit specified by this mask is set, potential line breaks may be located.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

Text Boundary Operation Class

Identifies the class of Unicode utility operations that find text boundaries.

```
enum {
    kUnicodeTextBreakClass = 'ubrk'
};
```

Constants

`kUnicodeTextBreakClass`
: Identifies the class of Unicode utility operations that find text boundaries.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeUtilities.h`.

Discussion

The locales and text-break variants available for finding boundaries in Unicode text can be determined by calling the Locales Utilities functions `LocaleOperationCountLocales` and `LocaleOperationGetLocales` with the `opClass` parameter set to the `kUnicodeTextBreakClass` constant.

Other References

Backup Core Reference

Framework: CoreServices/CoreServices.h

Declared in BackupCore.h

Overview

Backup Core is a C language API that supplies a low-level interface to file and folder settings used by Backup. Introduced in Mac OS X v10.5, Backup is a built-in, user-configurable backup solution that protects user data from accidental loss.

The Backup Core API includes a function you can use to exclude from Backup temporary or otherwise unimportant folders and files your application creates. In addition, you can use this function to allow your users to make backup decisions from within the context of your application.

Functions

CSBackupIsItemExcluded

Returns a Boolean value indicating whether an item is currently excluded from the backup.

```
Boolean CSBackupIsItemExcluded (
    CFURLRef item,
    Boolean * excludeByPath
);
```

Parameters

item

The URL of the item.

excludeByPath

If `true`, the item's backup exclusion status applies to its location; if `false`, the item's backup exclusion status applies to itself, regardless of its location. See `CSBackupSetItemExcluded` (page 2190) for more information. Can be `NULL`.

Return Value

`true` if the item or any of its ancestors are currently excluded from backup, `false` otherwise.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`BackupCore.h`

CSBackupSetItemExcluded

Includes or excludes an item from the backup.

```
OSStatus CSBackupSetItemExcluded (
    CFURLRef item,
    Boolean exclude,
    Boolean excludeByPath
);
```

Parameters

item

The URL of the file or folder to be included or excluded from the backup.

exclude

Pass `true` to exclude this item from backup (Backup will not back up this item). Pass `false` to stop excluding this item (Backup will back up this item if the user so chooses).

excludeByPath

Pass `true` to indicate that this item is excluded because of its location (its absolute path). Pass `false` to indicate that this item is excluded regardless of its location (and regardless of whether the user moves the item).

Return Value

A result code. Returns `noErr` if the item was successfully included or excluded from the backup.

Discussion

Backup skips files and folders marked for exclusion. If a folder is marked for exclusion, the folder and all its contents are excluded from the backup process. You can exclude specific paths that do not exist yet, but that you plan to create later, by passing the planned URL in `item` and passing `true` in `excludeByPath`. Note that if you pass `false` in `excludeByPath`, the URL must already exist.

Your application can allow users to change the backup exclusion status of any file or folder to which it has write access. To change the backup exclusion status of a path, your application must be running with administrator privileges.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`BackupCore.h`

Low Memory Accessors Reference

Framework: CoreServices/CoreServices.h

Declared in LowMem.h

Overview

Note: This document was previously titled *Carbon Specification for Low Memory Accessors*.

Low-memory accessors are functions you can use to obtain or update information stored in low-memory variables. In general, don't think of the values returned by low-memory accessors as residing in low memory—think of them as global information for a process, possibly associated with a specific manager or service, that is set or returned by Mac OS X. These functions allow you to access useful system information (such as the location of the mouse) stored as global variables in so-called "low memory."

The practice of accessing low memory directly was questionable in earlier versions of Mac OS, and certainly is not suggested now. In some cases, better, safer functions exist in other interfaces for obtaining the same information. To facilitate the porting of legacy applications, Carbon supports many of the low-memory accessors. However, you should always avoid using low-memory accessors if there are direct calls to obtain the same information.

Functions

LMGetApFontID

(Deprecated in Mac OS X v10.4. Use `GetAppFont` instead.)

```
SInt16 LMGetApFontID (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`LowMem.h`

LMGetBootDrive

```
SInt16 LMGetBootDrive (
   void
);
```

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

Declared In
LowMem.h

LMGetBufPtr

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
Ptr LMGetBufPtr (
   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
LowMem.h

LMGetBufTgDate

(Deprecated in Mac OS X v10.5.)

```
SInt32 LMGetBufTgDate (
   void
);
```

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
LowMem.h

LMGetBufTgFBkNum

(Deprecated in Mac OS X v10.5.)

```
SInt16 LMGetBufTgFBkNum (
   void
);
```

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
`LowMem.h`

LMGetBufTgFFlg

(Deprecated in Mac OS X v10.5.)

```
SInt16 LMGetBufTgFFlg (
   void
);
```

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
`LowMem.h`

LMGetBufTgFNum

(Deprecated in Mac OS X v10.5.)

```
SInt32 LMGetBufTgFNum (
   void
);
```

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
`LowMem.h`

LMGetCPUFlag

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
UInt8 LMGetCPUFlag (
   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
`LowMem.h`

LMGetCurApName

(Deprecated in Mac OS X v10.5.)

```
StringPtr LMGetCurApName (
   void
);
```

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
`LowMem.h`

LMGetCurApRefNum

(Deprecated in Mac OS X v10.5.)

```
FSIORefNum LMGetCurApRefNum (
   void
);
```

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
`LowMem.h`

LMGetCurPageOption

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
SInt16 LMGetCurPageOption (
   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
`LowMem.h`

LMGetCurPitch

(Deprecated in Mac OS X v10.5.)

```
SInt16 LMGetCurPitch (
   void
);
```

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
`LowMem.h`

LMGetCurStackBase

(Deprecated in Mac OS X v10.4. Use the Thread Manager function `ThreadCurrentStackSpace` (page 2117) or the POSIX threads function `pthread_get_stackaddr_np` instead.)

```
Ptr LMGetCurStackBase (
   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
`LowMem.h`

LMGetDefltStack

(Deprecated in Mac OS X v10.5.)

```
SInt32 LMGetDefltStack (
   void
);
```

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
`LowMem.h`

LMGetDiskFormatingHFSDefaults

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
Ptr LMGetDiskFormatingHFSDefaults (
   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
`LowMem.h`

LMGetFinderName

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
StringPtr LMGetFinderName (
   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
`LowMem.h`

LMGetGZMoveHnd

(Deprecated in Mac OS X v10.5.)

```
Handle LMGetGZMoveHnd (
   void
);
```

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
`LowMem.h`

LMGetGZRootHnd

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
Handle LMGetGZRootHnd (
   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
`LowMem.h`

LMGetHeapEnd

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
Ptr LMGetHeapEnd (
   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
`LowMem.h`

LMGetHighHeapMark

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
Ptr LMGetHighHeapMark (
   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
`LowMem.h`

LMGetIntlSpec

```
Ptr LMGetIntlSpec (
   void
);
```

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

Declared In
`LowMem.h`

LMGetJStash

(Deprecated in Mac OS X v10.5.)

```
UniversalProcPtr LMGetJStash (
   void
);
```

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
`LowMem.h`

LMGetLvl2DT

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
UniversalProcPtr LMGetLvl2DT (
   short vectorNumber
);
```

Parameters

vectorNumber

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
LowMem.h

LMGetMemTop

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
Ptr LMGetMemTop (
   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
LowMem.h

LMGetMinStack

(Deprecated in Mac OS X v10.5.)

```
SInt32 LMGetMinStack (
   void
);
```

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
LowMem.h

LMGetMinusOne

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
SInt32 LMGetMinusOne (
   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
`LowMem.h`

LMGetOneOne

(Deprecated in Mac OS X v10.5.)

```
SInt32 LMGetOneOne (
   void
);
```

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
`LowMem.h`

LMGetPrintErr

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
SInt16 LMGetPrintErr (
   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
`LowMem.h`

LMGetResErr

```
SInt16 LMGetResErr (
   void
);
```

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

Declared In
`LowMem.h`

LMGetResLoad

```
UInt8 LMGetResLoad (
   void
);
```

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

Declared In
`LowMem.h`

LMGetRndSeed

(Deprecated in Mac OS X v10.5.)

```
SInt32 LMGetRndSeed (
   void
);
```

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
`LowMem.h`

LMGetScrDmpEnb

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
UInt8 LMGetScrDmpEnb (
   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
LowMem.h

LMGetSdVolume

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
UInt8 LMGetSdVolume (
   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
LowMem.h

LMGetSEvtEnb

Returns a value that specifies the system event enabled bit. (Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
UInt8 LMGetSEvtEnb (
   void
);
```

Discussion
`LMGetSEvtEnb` returns a signed 16-bit integer that describes the low-memory system event enabled bit, a byte that, if set to 0, causes `SystemEvent` to always return `false`.

The value obtained by `LMGetSEvtEnb` is also accessible in the system global variable `SEvtEnb`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
LowMem.h

LMGetSoundBase

(Deprecated in Mac OS X v10.5.)

```
Ptr LMGetSoundBase (
   void
);
```

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
`LowMem.h`

LMGetSoundLevel

(Deprecated in Mac OS X v10.5.)

```
UInt8 LMGetSoundLevel (
   void
);
```

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
`LowMem.h`

LMGetSoundPtr

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
Ptr LMGetSoundPtr (
   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
`LowMem.h`

LMGetStackLowPoint

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
Ptr LMGetStackLowPoint (
   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
`LowMem.h`

LMGetSysFontFam

(Deprecated in Mac OS X v10.4. Use `GetSysFont` instead.)

```
SInt16 LMGetSysFontFam (
   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
`LowMem.h`

LMGetSysFontSize

(Deprecated in Mac OS X v10.4. Use `GetDefFontSize` instead.)

```
SInt16 LMGetSysFontSize (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`LowMem.h`

LMGetSysMap

```
SInt16 LMGetSysMap (
   void
);
```

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

Declared In
LowMem.h

LMGetSysResName

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
StringPtr LMGetSysResName (
   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
LowMem.h

LMGetTmpResLoad

```
UInt8 LMGetTmpResLoad (
   void
);
```

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

Declared In
LowMem.h

LMGetToExtFS

(Deprecated in Mac OS X v10.5.)

```
UniversalProcPtr LMGetToExtFS (
   void
);
```

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
LowMem.h

LMGetToolScratch

(Deprecated in Mac OS X v10.4. Use process global data instead.)

```
Ptr LMGetToolScratch (
   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
LowMem.h

LMSetApFontID

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetApFontID (
   SInt16 value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
LowMem.h

LMSetBootDrive

```
void LMSetBootDrive (
   SInt16 value
);
```

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

Declared In
LowMem.h

LMSetBufPtr

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetBufPtr (
   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
`LowMem.h`

LMSetBufTgDate

(Deprecated in Mac OS X v10.5.)

```
void LMSetBufTgDate (
   SInt32 value
);
```

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
`LowMem.h`

LMSetBufTgFBkNum

(Deprecated in Mac OS X v10.5.)

```
void LMSetBufTgFBkNum (
   SInt16 value
);
```

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
`LowMem.h`

LMSetBufTgFFlg

(Deprecated in Mac OS X v10.5.)

```
void LMSetBufTgFFlg (
   SInt16 value
);
```

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
`LowMem.h`

LMSetBufTgFNum

(Deprecated in Mac OS X v10.5.)

```
void LMSetBufTgFNum (
   SInt32 value
);
```

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
`LowMem.h`

LMSetCPUFlag

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetCPUFlag (
   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
`LowMem.h`

LMSetCurApName

(Deprecated in Mac OS X v10.5.)

```
void LMSetCurApName (
   ConstStr31Param curApNameValue
);
```

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
`LowMem.h`

LMSetCurApRefNum

(Deprecated in Mac OS X v10.5.)

```
void LMSetCurApRefNum (
   FSIORefNum value
);
```

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
`LowMem.h`

LMSetCurPageOption

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetCurPageOption (
   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
`LowMem.h`

LMSetCurPitch

(Deprecated in Mac OS X v10.5.)

```
void LMSetCurPitch (
   SInt16 value
);
```

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
`LowMem.h`

LMSetCurStackBase

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetCurStackBase (
   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
`LowMem.h`

LMSetDefltStack

(Deprecated in Mac OS X v10.5.)

```
void LMSetDefltStack (
   SInt32 value
);
```

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
`LowMem.h`

LMSetDiskFormatingHFSDefaults

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetDiskFormatingHFSDefaults (
   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
`LowMem.h`

LMSetFinderName

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetFinderName (
   ConstStr15Param finderNameValue
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`LowMem.h`

LMSetGZMoveHnd

(Deprecated in Mac OS X v10.5.)

```
void LMSetGZMoveHnd (
   Handle value
);
```

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
`LowMem.h`

LMSetGZRootHnd

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetGZRootHnd (
   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
`LowMem.h`

LMSetHeapEnd

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetHeapEnd (
   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
`LowMem.h`

LMSetHighHeapMark

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetHighHeapMark (
   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
`LowMem.h`

LMSetIntlSpec

```
void LMSetIntlSpec (
   Ptr value
);
```

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

Declared In
LowMem.h

LMSetJStash

(Deprecated in Mac OS X v10.5.)

```
void LMSetJStash (
   UniversalProcPtr value
);
```

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
LowMem.h

LMSetLvl2DT

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetLvl2DT (
   UniversalProcPtr Lvl2DTValue,
   short vectorNumber
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
LowMem.h

LMSetMemTop

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetMemTop (
   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
`LowMem.h`

LMSetMinStack

(Deprecated in Mac OS X v10.5.)

```
void LMSetMinStack (
   SInt32 value
);
```

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
`LowMem.h`

LMSetMinusOne

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetMinusOne (
   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
`LowMem.h`

LMSetOneOne

(Deprecated in Mac OS X v10.5.)

```
void LMSetOneOne (
   SInt32 value
);
```

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
`LowMem.h`

LMSetPrintErr

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetPrintErr (
   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
`LowMem.h`

LMSetResErr

```
void LMSetResErr (
   SInt16 value
);
```

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

Declared In
`LowMem.h`

LMSetResLoad

```
void LMSetResLoad (
   UInt8 value
);
```

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

Declared In
LowMem.h

LMSetRndSeed

(Deprecated in Mac OS X v10.5.)

```
void LMSetRndSeed (
   SInt32 value
);
```

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
LowMem.h

LMSetScrDmpEnb

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetScrDmpEnb (
   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
LowMem.h

LMSetSdVolume

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetSdVolume (
   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
LowMem.h

LMSetSEvtEnb

Sets the low-memory system event enabled bit. (Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetSEvtEnb (
   UInt8 value
);
```

Parameters

value

An unsigned 8-bit integer that describes the value of the system event enabled bit.

Discussion

`LMSetSEvtEnb` specifies an unsigned 8-bit integer that sets the low-memory system event enabled bit, a byte that, if set to 0, causes the `SystemEvent` to always return `false`.

The value set by `LMSetSEvtEnb` is also accessible in the system global variable `SEvtEnb`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`LowMem.h`

LMSetSoundBase

(Deprecated in Mac OS X v10.5.)

```
void LMSetSoundBase (
   Ptr value
);
```

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

`LowMem.h`

LMSetSoundLevel

(Deprecated in Mac OS X v10.5.)

```
void LMSetSoundLevel (
   UInt8 value
);
```

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
LowMem.h

LMSetSoundPtr

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetSoundPtr (
   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
LowMem.h

LMSetStackLowPoint

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetStackLowPoint (
   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
LowMem.h

LMSetSysFontFam

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetSysFontFam (
   SInt16 value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
LowMem.h

LMSetSysFontSize

```
void LMSetSysFontSize (
   SInt16 value
);
```

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

Declared In
LowMem.h

LMSetSysMap

```
void LMSetSysMap (
   SInt16 value
);
```

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

Declared In
LowMem.h

LMSetSysResName

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetSysResName (
   ConstStr15Param sysResNameValue
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
LowMem.h

LMSetTmpResLoad

```
void LMSetTmpResLoad (
   UInt8 value
);
```

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

Declared In
LowMem.h

LMSetToExtFS

(Deprecated in Mac OS X v10.5.)

```
void LMSetToExtFS (
   UniversalProcPtr value
);
```

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
LowMem.h

LMSetToolScratch

(Deprecated in Mac OS X v10.4. Use process global data instead.)

```
void LMSetToolScratch (
   const void *toolScratchValue
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
LowMem.h

Core Endian Reference

Framework: CoreServices/CoreServices.h
Declared in Endian.h

Overview

Core Endian Reference provides routines for converting data between big endian and little endian format. These routines are useful for developers who write code that must compile for multiple architectures, including:

- Macintosh developers who want to produce a universal binary.
- QuickTime developers who want their code to run in Windows as well as in Mac OS X.

Both QuickTime and Macintosh developers can use this API to handle reading or writing data to a file or network packet.

Macintosh developers can use this API to create and install callbacks that are invoked by Mac OS X automatically when your application:

- reads and writes custom resource data
- sends or receives custom Apple events
- reads and writes custom pasteboard data

The functions in this API are designed to do nothing when the target runtime is already in the desired format.

For more information see:

- Universal Binary Programming Guidelines
- QuickTime API Reference

Functions by Task

Working With Flippers

`CoreEndianInstallFlipper` (page 2226)
Installs a flipper callback for the specified data type.

`CoreEndianGetFlipper` (page 2225)
Obtains the flipper callback that is installed for the specified data type.

`CoreEndianFlipData` (page 2224)
Calls the flipper callback associated with the specified data type.

Changing the Endian Format

`Endian16_Swap` (page 2227)
Changes the endian format of an unsigned 16-bit integer.

`Endian32_Swap` (page 2227)
Changes the endian format of an unsigned 32-bit integer.

`Endian64_Swap` (page 2227)
Changes the endian format of an unsigned 64-bit integer.

Converting from Big-Endian to Native Format

`EndianS16_BtoN` (page 2228)
Converts a signed 16-bit big-endian value to the equivalent value in the computer's native format.

`EndianS32_BtoN` (page 2231)
Converts a signed 32-bit big-endian value to the equivalent value in the computer's native format.

`EndianS64_BtoN` (page 2233)
Converts a signed 64-bit big-endian value to the equivalent value in the computer's native format.

`EndianU16_BtoN` (page 2236)
Converts an unsigned 16-bit big-endian value to the equivalent value in the computer's native format.

`EndianU32_BtoN` (page 2238)
Converts an unsigned 32-bit big-endian value to the equivalent value in the computer's native format.

`EndianU64_BtoN` (page 2240)
Converts an unsigned 64-bit big-endian value to the equivalent value in the computer's native format.

Converting from Native Format to Big-Endian Format

`EndianS16_NtoB` (page 2229)
Converts a signed 16-bit value in the computer's native format to the equivalent big-endian value.

`EndianS32_NtoB` (page 2232)
Converts a signed 32-bit value in the computer's native format to the equivalent big-endian value.

`EndianS64_NtoB` (page 2234)
Converts a signed 64-bit value in the computer's native format to the equivalent big-endian value.

`EndianU16_NtoB` (page 2237)
Converts an unsigned 16-bit value in the computer's native format to the equivalent big-endian value.

`EndianU32_NtoB` (page 2239)
Converts an unsigned 32-bit value in the computer's native format to the equivalent big-endian value.

`EndianU64_NtoB` (page 2242)
Converts an unsigned 64-bit value in the computer's native format to the equivalent big-endian value.

Converting from Little-Endian Format to Native Format

`EndianS16_LtoN` (page 2229)
Converts a signed 16-bit little-endian value to the equivalent value in the computer's native format.

`EndianS32_LtoN` (page 2232)
Converts a signed 32-bit little-endian value to the equivalent value in the computer's native format.

`EndianS64_LtoN` (page 2234)
Converts a signed 64-bit little-endian value to the equivalent value in the computer's native format.

`EndianU16_LtoN` (page 2236)
Converts an unsigned 16-bit little-endian value to the equivalent value in the computer's native format.

`EndianU32_LtoN` (page 2239)
Converts an unsigned 32-bit little-endian value to the equivalent value in the computer's native format.

`EndianU64_LtoN` (page 2241)
Converts an unsigned 64-bit little-endian value to the equivalent value in the computer's native format.

Converting from Native Format to Little-Endian Format

`EndianS16_NtoL` (page 2230)
Converts a signed 16-bit value in the computer's native format to the equivalent little-endian value.

`EndianS32_NtoL` (page 2232)
Converts a signed 32-bit value in the computer's native format to the equivalent little-endian value.

`EndianS64_NtoL` (page 2235)
Converts a signed 64-bit value in the computer's native format to the equivalent little-endian value.

`EndianU16_NtoL` (page 2237)
Converts an unsigned 16-bit value in the computer's native format to the equivalent little-endian value.

`EndianU32_NtoL` (page 2240)
Converts an unsigned 32-bit value in the computer's native format to the equivalent little-endian value.

`EndianU64_NtoL` (page 2242)
Converts an unsigned 64-bit value in the computer's native format to the equivalent little-endian value.

Converting from Big-Endian to Little-Endian Format

`EndianS16_BtoL` (page 2228)
Converts a signed 16-bit big-endian value to the equivalent little-endian value.

`EndianS32_BtoL` (page 2230)
Converts a signed 32-bit big-endian value to the equivalent little-endian value.

`EndianS64_BtoL` (page 2233)
Converts a signed 64-bit big-endian value to the equivalent little-endian value.

`EndianU16_BtoL` (page 2235)
Converts an unsigned 16-bit big-endian value to the equivalent little-endian value.

`EndianU32_BtoL` (page 2238)
Converts an unsigned 32-bit big-endian value to the equivalent little-endian value.

`EndianU64_BtoL` (page 2240)
Converts an unsigned 64-bit big-endian value to the equivalent little-endian value.

Converting From Little-Endian to Big-Endian Format

`EndianS16_LtoB` (page 2229)
Converts a signed 16-bit little-endian value to the equivalent big-endian value.

`EndianS32_LtoB` (page 2231)
Converts a signed 32-bit little-endian value to the equivalent big-endian value.

`EndianS64_LtoB` (page 2234)
Converts a signed 64-bit little-endian value to the equivalent big-endian value.

`EndianU16_LtoB` (page 2236)
Converts an unsigned 16-bit little-endian value to the equivalent big-endian value.

`EndianU32_LtoB` (page 2238)
Converts an unsigned 32-bit little-endian value to the equivalent big-endian value.

`EndianU64_LtoB` (page 2241)
Converts an unsigned 64-bit little-endian value to the equivalent big-endian value.

Functions

CoreEndianFlipData

Calls the flipper callback associated with the specified data type.

```
OSStatus CoreEndianFlipData (
   OSType dataDomain,
   OSType dataType,
   SInt16 id,
   void *data,
   ByteCount dataLen,
   Boolean currentlyNative
);
```

Parameters

dataDomain
An `OSType` value that specifies the domain of the flipper callback you want to invoke. Pass `kCoreEndianResourceManagerDomain` (page 2248) if your callback applies to resource data. Pass `kCoreEndianAppleEventManagerDomain` (page 2248) if your callback applies to Apple event data. See "Domain Types" (page 2247) for more information.

dataType

An `OSType` value that specifies the type of data that needs to be byte-swapped. This is the four character code of the resource type or Apple event. This never needs to be byte-swapped even though GDB and Xcode display the resource in byte-swapped order.

id

The resource ID of the data type. The Resource Manager byte-swaps this for you so you can compare the resource ID against constants in your code. If the data is not a resource, pass `0`.

data

A pointer to the first byte of the data to be byte swapped.

dataLen

The length of the data (in bytes) to be byte swapped.

currentlyNative

A Boolean value that indicates the direction to byte swap. Pass `true` when the data specified by the `data` parameter uses the byte ordering of the currently executing code. On a PowerPC system, `true` specifies that the data is in big-endian format. On an x86 system, `true` specifies that the data is in little-endian format.

Return Value

A result code. Returns `noErr` if the data is byte swapped and `handlerNotFound` if the data is not byte swapped. Note that data is only byte swapped if it needs to be byte swapped.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`Endian.h`

CoreEndianGetFlipper

Obtains the flipper callback that is installed for the specified data type.

```
OSStatus CoreEndianGetFlipper (
   OSType dataDomain,
   OSType dataType,
   CoreEndianFlipProc *proc,
   void **refcon
);
```

Parameters

dataDomain

An `OSType` value that specifies the domain of the flipper callback you want to obtain. Pass `kCoreEndianResourceManagerDomain` (page 2248) to obtain a callback that applies to resource data. Pass `kCoreEndianAppleEventManagerDomain` (page 2248) to obtain a callback that applies to Apple event data. See "Domain Types" (page 2247) for more information.

dataType

An `OSType` value that specifies the type of data associated with the flipper callback you want to obtain. This is the four character code of the resource type or Apple event. This never needs to be byte-swapped even though GDB and Xcode display the resource in byte-swapped order.

proc

On output, points to the flipper callback that is installed for the data type specified by the `dataType` parameter.

refCon
On output, points to a 32-bit value that references callback-specific data.

Return Value
A result code. Returns `noErr` if the flipper callback is found.

Discussion
You can call the function `CoreEndianGetFlipper` to determine whether a flipper for a given data type is available.

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

Declared In
`Endian.h`

CoreEndianInstallFlipper

Installs a flipper callback for the specified data type.

```
OSStatus CoreEndianInstallFlipper (
   OSType dataDomain,
   OSType dataType,
   CoreEndianFlipProc proc,
   void *refcon
);
```

Parameters

dataDomain
An `OSType` value that specifies the domain to which the flipper callback applies. Pass `kCoreEndianResourceManagerDomain` (page 2248) if your callback applies to resource data. Pass `kCoreEndianAppleEventManagerDomain` (page 2248) if your callback applies to Apple event data. See "Domain Types" (page 2247) for more information.

dataType
An `OSType` value that specifies the type of data for which you want your flipper callback installed. This is the four character code of the resource type or Apple event.

proc
A pointer to your flipper callback. The flipper callback is installed into a per-process table that is searched before the system table.

refCon
A 32-bit value containing or referring to data needed by the callback.

Return Value
A result code. Returns `noErr` if your flipper callback is installed.

Discussion
You should install the callback by calling the function `CoreEndianInstallFlipper` when your application calls its initialization routine or when you open your resource file.

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

Declared In
`Endian.h`

Endian16_Swap

Changes the endian format of an unsigned 16-bit integer.

```
UInt16 Endian16_Swap (
    UInt16    value
);
```

Parameters

value

An unsigned 16-bit integer input.

Return Value

The unsigned 16-bit integer result.

Availability

Available in Mac OS X v10.0 and later.

Available in QuickTime 5 and later for Windows.

Declared In

`Endian.h`

Endian32_Swap

Changes the endian format of an unsigned 32-bit integer.

```
UInt32 Endian32_Swap (
    UInt32    value
);
```

Parameters

value

An unsigned 32-bit integer input.

Return Value

The unsigned 32-bit integer result.

Availability

Available in Mac OS X v10.0 and later.

Available in QuickTime 5 and later for Windows.

Declared In

`Endian.h`

Endian64_Swap

Changes the endian format of an unsigned 64-bit integer.

```
static UInt64 Endian64_Swap (
   UInt64 value
);
```

Parameters

value

An unsigned 64-bit integer input.

Return Value
The unsigned 64-bit integer result.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 5 and later for Windows.

Declared In
`Endian.h`

EndianS16_BtoL

Converts a signed 16-bit big-endian value to the equivalent little-endian value.

```
SInt16 EndianS16_BtoL (
    SInt16    value
);
```

Parameters

value
A signed 16-bit big-endian value.

Return Value
The equivalent little-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianS16_BtoN

Converts a signed 16-bit big-endian value to the equivalent value in the computer's native format.

```
SInt16 EndianS16_BtoN (
    SInt16    value
);
```

Parameters

value
A signed 16-bit big-endian value.

Return Value
The equivalent value in the computer's native format.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Related Sample Code
QTCarbonShell

Declared In
Endian.h

EndianS16_LtoB

Converts a signed 16-bit little-endian value to the equivalent big-endian value.

```
SInt16 EndianS16_LtoB (
    SInt16    value
);
```

Parameters

value
A signed 16-bit little-endian value.

Return Value
The equivalent big-endian value.

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

Declared In
Endian.h

EndianS16_LtoN

Converts a signed 16-bit little-endian value to the equivalent value in the computer's native format.

```
SInt16 EndianS16_LtoN (
    SInt16    value
);
```

Parameters

value
A signed 16-bit little-endian value.

Return Value
The equivalent value in the computer's native format.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
Endian.h

EndianS16_NtoB

Converts a signed 16-bit value in the computer's native format to the equivalent big-endian value.

```
SInt16 EndianS16_NtoB (
    SInt16    value
);
```

Parameters

value

A signed 16-bit value in the computer's native format.

Return Value
The equivalent big-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianS16_NtoL

Converts a signed 16-bit value in the computer's native format to the equivalent little-endian value.

```
SInt16 EndianS16_NtoL (
    SInt16    value
);
```

Parameters

value

A signed 16-bit value in the computer's native format.

Return Value
The equivalent little-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianS32_BtoL

Converts a signed 32-bit big-endian value to the equivalent little-endian value.

```
SInt32 EndianS32_BtoL (
    SInt32    value
);
```

Parameters

value

A signed 32-bit big-endian value.

Return Value
The equivalent little-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianS32_BtoN

Converts a signed 32-bit big-endian value to the equivalent value in the computer's native format.

```
SInt32 EndianS32_BtoN (
    SInt32    value
);
```

Parameters

value
A signed 32-bit big-endian value.

Return Value
The equivalent value in the computer's native format.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianS32_LtoB

Converts a signed 32-bit little-endian value to the equivalent big-endian value.

```
SInt32 EndianS32_LtoB (
    SInt32    value
);
```

Parameters

value
A signed 32-bit little-endian value.

Return Value
The equivalent big-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianS32_LtoN

Converts a signed 32-bit little-endian value to the equivalent value in the computer's native format.

```
SInt32 EndianS32_LtoN (
    SInt32    value
);
```

Parameters

value

A signed 32-bit little-endian value.

Return Value

The equivalent value in the computer's native format.

Availability

Available in Mac OS X v10.0 and later.

Available in QuickTime 4 and later for Windows.

Declared In

`Endian.h`

EndianS32_NtoB

Converts a signed 32-bit value in the computer's native format to the equivalent big-endian value.

```
SInt32 EndianS32_NtoB (
    SInt32    value
);
```

Parameters

value

A signed 32-bit value in the computer's native format.

Return Value

The equivalent big-endian value.

Availability

Available in Mac OS X v10.0 and later.

Available in QuickTime 4 and later for Windows.

Declared In

`Endian.h`

EndianS32_NtoL

Converts a signed 32-bit value in the computer's native format to the equivalent little-endian value.

```
SInt32 EndianS32_NtoL (
    SInt32    value
);
```

Parameters

value

A signed 32-bit value in the computer's native format.

Return Value
The equivalent little-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianS64_BtoL

Converts a signed 64-bit big-endian value to the equivalent little-endian value.

```
SInt64 EndianS64_BtoL (
    SInt64    value
);
```

Parameters

value
A signed 64-bit big-endian value.

Return Value
The equivalent little-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianS64_BtoN

Converts a signed 64-bit big-endian value to the equivalent value in the computer's native format.

```
SInt64 EndianS64_BtoN (
    SInt64    value
);
```

Parameters

value
A signed 64-bit big-endian value.

Return Value
The equivalent value in the computer's native format.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianS64_LtoB

Converts a signed 64-bit little-endian value to the equivalent big-endian value.

```
SInt64 EndianS64_LtoB (
    SInt64    value
);
```

Parameters

value

A signed 64-bit little-endian value.

Return Value

The equivalent big-endian value.

Availability

Available in Mac OS X v10.0 and later.

Available in QuickTime 4 and later for Windows.

Declared In

`Endian.h`

EndianS64_LtoN

Converts a signed 64-bit little-endian value to the equivalent value in the computer's native format.

```
SInt64 EndianS64_LtoN (
    SInt64    value
);
```

Parameters

value

A signed 64-bit little-endian value.

Return Value

The equivalent value in the computer's native format.

Availability

Available in Mac OS X v10.0 and later.

Available in QuickTime 4 and later for Windows.

Declared In

`Endian.h`

EndianS64_NtoB

Converts a signed 64-bit value in the computer's native format to the equivalent big-endian value.

```
SInt64 EndianS64_NtoB (
    SInt64    value
);
```

Parameters

value

A signed 64-bit value in the computer's native format.

Return Value
The equivalent big-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianS64_NtoL

Converts a signed 64-bit value in the computer's native format to the equivalent little-endian value.

```
SInt64 EndianS64_NtoL (
    SInt64    value
);
```

Parameters

value
A signed 64-bit value in the computer's native format.

Return Value
The equivalent little-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianU16_BtoL

Converts an unsigned 16-bit big-endian value to the equivalent little-endian value.

```
UInt16 EndianU16_BtoL (
    UInt16    value
);
```

Parameters

value
An unsigned 16-bit big-endian value.

Return Value
The equivalent little-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianU16_BtoN

Converts an unsigned 16-bit big-endian value to the equivalent value in the computer's native format.

```
UInt16 EndianU16_BtoN (
    UInt16    value
);
```

Parameters

value

An unsigned 16-bit big-endian value.

Return Value

The equivalent value in the computer's native format.

Availability

Available in Mac OS X v10.0 and later.

Available in QuickTime 4 and later for Windows.

Declared In

`Endian.h`

EndianU16_LtoB

Converts an unsigned 16-bit little-endian value to the equivalent big-endian value.

```
UInt16 EndianU16_LtoB (
    UInt16    value
);
```

Parameters

value

An unsigned 16-bit little-endian value.

Return Value

The equivalent big-endian value.

Availability

Available in Mac OS X v10.0 and later.

Available in QuickTime 4 and later for Windows.

Declared In

`Endian.h`

EndianU16_LtoN

Converts an unsigned 16-bit little-endian value to the equivalent value in the computer's native format.

```
UInt16 EndianU16_LtoN (
    UInt16    value
);
```

Parameters

value

An unsigned 16-bit little-endian value.

Return Value
The equivalent value in the computer's native format.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianU16_NtoB

Converts an unsigned 16-bit value in the computer's native format to the equivalent big-endian value.

```
UInt16 EndianU16_NtoB (
    UInt16    value
);
```

Parameters

value
An unsigned 16-bit value in the computer's native format.

Return Value
The equivalent big-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianU16_NtoL

Converts an unsigned 16-bit value in the computer's native format to the equivalent little-endian value.

```
UInt16 EndianU16_NtoL (
    UInt16    value
);
```

Parameters

value
An unsigned 16-bit value in the computer's native format.

Return Value
The equivalent little-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianU32_BtoL

Converts an unsigned 32-bit big-endian value to the equivalent little-endian value.

```
UInt32 EndianU32_BtoL (
    UInt32    value
);
```

Parameters

value

An unsigned 32-bit big-endian value.

Return Value

The equivalent little-endian value.

Availability

Available in Mac OS X v10.0 and later.

Available in QuickTime 4 and later for Windows.

Declared In

`Endian.h`

EndianU32_BtoN

Converts an unsigned 32-bit big-endian value to the equivalent value in the computer's native format.

```
UInt32 EndianU32_BtoN (
    UInt32    value
);
```

Parameters

value

An unsigned 32-bit big-endian value.

Return Value

The equivalent value in the computer's native format.

Availability

Available in Mac OS X v10.0 and later.

Available in QuickTime 4 and later for Windows.

Declared In

`Endian.h`

EndianU32_LtoB

Converts an unsigned 32-bit little-endian value to the equivalent big-endian value.

```
UInt32 EndianU32_LtoB (
    UInt32    value
);
```

Parameters

value

An unsigned 32-bit little-endian value.

Return Value
The equivalent big-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianU32_LtoN

Converts an unsigned 32-bit little-endian value to the equivalent value in the computer's native format.

```
UInt32 EndianU32_LtoN (
    UInt32    value
);
```

Parameters

value
An unsigned 32-bit little-endian value.

Return Value
The equivalent value in the computer's native format.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianU32_NtoB

Converts an unsigned 32-bit value in the computer's native format to the equivalent big-endian value.

```
UInt32 EndianU32_NtoB (
    UInt32    value
);
```

Parameters

value
An unsigned 32-bit value in the computer's native format.

Return Value
The equivalent big-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Related Sample Code
QTMetaData

Declared In
`Endian.h`

EndianU32_NtoL

Converts an unsigned 32-bit value in the computer's native format to the equivalent little-endian value.

```
UInt32 EndianU32_NtoL (
    UInt32    value
);
```

Parameters

value

An unsigned 32-bit value in the computer's native format.

Return Value
The equivalent little-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianU64_BtoL

Converts an unsigned 64-bit big-endian value to the equivalent little-endian value.

```
UInt64 EndianU64_BtoL (
    UInt64    value
);
```

Parameters

value

An unsigned 64-bit big-endian value.

Return Value
The equivalent little-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianU64_BtoN

Converts an unsigned 64-bit big-endian value to the equivalent value in the computer's native format.

```
UInt64 EndianU64_BtoN (
    UInt64    value
);
```

Parameters

value

An unsigned 64-bit big-endian value.

Return Value
The equivalent value in the computer's native format.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianU64_LtoB

Converts an unsigned 64-bit little-endian value to the equivalent big-endian value.

```
UInt64 EndianU64_LtoB (
    UInt64    value
);
```

Parameters

value

An unsigned 64-bit little-endian value.

Return Value
The equivalent big-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianU64_LtoN

Converts an unsigned 64-bit little-endian value to the equivalent value in the computer's native format.

```
UInt64 EndianU64_LtoN (
    UInt64    value
);
```

Parameters

value

An unsigned 64-bit little-endian value.

Return Value
The equivalent value in the computer's native format.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianU64_NtoB

Converts an unsigned 64-bit value in the computer's native format to the equivalent big-endian value.

```
UInt64 EndianU64_NtoB (
    UInt64    value
);
```

Parameters

value
An unsigned 64-bit value in the computer's native format.

Return Value
The equivalent big-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

EndianU64_NtoL

Converts an unsigned 64-bit value in the computer's native format to the equivalent little-endian value.

```
UInt64 EndianU64_NtoL (
    UInt64    value
);
```

Parameters

value
An unsigned 64-bit value in the computer's native format.

Return Value
The equivalent little-endian value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

Callbacks

CoreEndianFlipProc

Defines a pointer to a callback function that byte-swaps your custom data.

```
typedef CALLBACK_API (OSStatus, CoreEndianFlipProc)
   (OSType dataDomain,
   OSType dataType,
   short id,
   void *dataPtr,
   UInt32 dataSize,
   Boolean currentlyNative,
   void *refcon
);
```

You would declare your flipper callback function as follows if your were to name it `MyCoreEndianFlipProc`:

```
OSStatus MyCoreEndianFlipProc (
   OSType dataDomain,
   OSType dataType,
   short id,
   void *dataPtr,
   UInt32 dataSize,
   Boolean currentlyNative,
   void *refcon
);
```

Parameters

dataDomain

An `OSType` value that specifies the domain to which the flipper callback applies. The value `kCoreEndianResourceManagerDomain` signifies the domain is resource data. The value `kCoreEndianAppleEventManagerDomain` signifies the domain is Apple event data. See "Domain Types" (page 2247) for more information on the values that can be passed to your callback.

dataType

The type of data to be byte swapped by the callback. This is the four character code of the resource type or Apple event.

id

The resource id of the data type. The value `0` signifies the data is not a resource.

dataPtr

On input, points to the data to be flipped. On output, points to the byte-swapped data.

dataSize

The size of the data pointed to the by the `dataPtr` parameter.

currentlyNative

A Boolean value that indicates the direction to byte swap. The value `true` specifies the data pointed to by the `dataPtr` parameter uses the byte ordering of the currently executing code. On a PowerPC system, `true` specifies that the data is in big-endian format. On an x86 system, `true` specifies that the data is in little-endian format.

refcon

A 32-bit value that contains or refers to data needed by the callback.

Return Value
A result code that indicates whether the byte swapping is successful. Your callback should return `noErr` if the resource is byte swapped without error, `handlerNotFound` if you chose not to byte swap the data, and the appropriate result code to indicate an error condition if the data is bad. The result code you return is propagated through the appropriate manager (Resource Manager (`ResError`) or Apple Event Manager) to the caller.

Discussion
You should write each flipper callback so it traverses the data structure that contains the data and performs the following tasks:

- Byte swaps all Resource Manager counts and lengths so that array indexes are associated with the appropriate value
- Byte swaps all integers and longs so that when you read them into variables of a compatible type the values can be operated on correctly (such as numerical, offset, and shift operations)

A flipper callback must be bidirectional because it can be called by the Resource Manager or Apple Event Manager when you read data as well as when you write data. The system ensures that your flipper callback is invoked at the appropriate times.

Your flipper callback is not invoked on a microprocessor that uses big-endian byte ordering. It is called with `currentlyNative` set to `false` when data is read (or received) and `true` when the data is set to be written (or sent).

Availability
Available in Mac OS X 10.3 and later.

Declared In
`Endian.h`

Data Types

BigEndianLong

Protects a big-endian long value from being changed by little-endian code.

```
// Little-endian host
struct BigEndianLong {
long bigEndianValue;
};
typedef struct BigEndianLong  BigEndianLong;
// Big-endian host
typedef long  BigEndianLong;
```

Fields
`bigEndianValue`
A long value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

BigEndianUnsignedLong

Protects a big-endian unsigned long value from being changed by little-endian code.

```
// Little-endian host
struct BigEndianUnsignedLong {
 unsigned long    bigEndianValue;
 };
typedef struct BigEndianUnsignedLong BigEndianUnsignedLong;
// Big-endian host
typedef unsigned long  BigEndianUnsignedLong;
```

Fields
`bigEndianValue`
An unsigned long value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

BigEndianShort

Protects a big-endian short value from being changed by little-endian code.

```
// Little-endian host
struct BigEndianShort {
 short    bigEndianValue;
};
typedef struct BigEndianShort  BigEndianShort;
// Big-endian host
typedef short BigEndianShort;
```

Fields
`bigEndianValue`
A short value.

Availability
Available in Mac OS X v10.0 and later.
Available in QuickTime 4 and later for Windows.

Declared In
`Endian.h`

BigEndianUnsignedShort

Protects a big-endian unsigned short value from being changed by little-endian code.

```
// Little-endian host
struct BigEndianUnsignedShort {
 unsigned short    bigEndianValue;
};
typedef struct BigEndianUnsignedShort  BigEndianUnsignedShort;
// Big-endian host
typedef unsigned short BigEndianUnsignedShort;
```

Fields

`bigEndianValue`

An unsigned short value.

Availability

Available in Mac OS X v10.0 and later.

Available in QuickTime 4 and later for Windows.

Declared In

`Endian.h`

BigEndianFixed

Protects a big-endian Fixed value from being changed by little-endian code.

```
// Little-endian host
struct BigEndianFixed {
Fixed    bigEndianValue;
};
typedef struct BigEndianFixed  BigEndianFixed;
// Big-endian host
typedef Fixed BigEndianFixed;
```

Fields

`bigEndianValue`

A fixed value.

Availability

Available in Mac OS X v10.0 and later.

Available in QuickTime 4 and later for Windows.

Declared In

`Endian.h`

BigEndianUnsignedFixed

Protects a big-endian unsigned Fixed value from being changed by little-endian code.

```
// Little-endian host
struct BigEndianUnsignedFixed {
UnsignedFixed    bigEndianValue;
};
typedef struct BigEndianUnsignedFixed  BigEndianUnsignedFixed;
// Big-endian host
typedef UnsignedFixed BigEndianUnsignedFixed;
```

Fields

`bigEndianValue`

An unsigned fixed value.

Availability

Available in Mac OS X v10.0 and later.

Available in QuickTime 4 and later for Windows.

Declared In

`Endian.h`

BigEndianOSType

Protects a big-endian OSType value from being changed by little-endian code.

```
// Little-endian host
struct BigEndianOSType {
OSType    bigEndianValue;
};
typedef struct BigEndianOStype  BigEndianOStype;
// Big-endian host
typedef OSType BigEndianOSType;
```

Fields

`bigEndianValue`

An OSType value.

Availability

Available in Mac OS X v10.0 and later.

Available in QuickTime 4 and later for Windows.

Constants

Domain Types

Specify the domain to which a flipper callback should be applied.

```
enum {
    kCoreEndianResourceManagerDomain = 'rsrc',
    kCoreEndianAppleEventManagerDomain = 'aevt'
};
```

Constants

`kCoreEndianResourceManagerDomain`

Specifies that the domain is limited to the resources for a specific application.

Available in Mac OS X v10.4 and later.

Declared in `Endian.h`.

`kCoreEndianAppleEventManagerDomain`

Specifies that the domain is limited to Apple events.

Available in Mac OS X v10.4 and later.

Declared in `Endian.h`.

Discussion

The data types have specific meanings within their domain, although some data types can be registered with the same callback in several domains.

Availability

Available in Mac OS X 10.3 and later.

Error Handler Reference

Framework: CoreServices/CoreServices.h

Declared in MacErrors.h
MacTypes.h

Overview

In Mac OS 9, Error Handler provides a system service that displays system alerts seen during startup, and assumes control when certain low-level errors occur. For detailed information, see the chapter "System Error Handler" in *Inside Macintosh: Operating System Utilities*. A PDF version of this chapter is available at http://developer.apple.com/documentation/mac/pdf/Operating_System_Utilities/pdf.html.

Note: In Mac OS X, Error Handler is no longer used for this purpose—for more information, see `SysError` (page 2249).

The header `MacErrors.h` defines result codes that are used by many Carbon functions to indicate their return status. Developers can also define custom result codes for their own applications, using the range 1000 through 9999 inclusive (Apple reserves all values outside of this range for internal use.) For more information, see Technical Q&A OV02 at http://developer.apple.com/qa/ov/ov02.html.

Functions

SysError

In Mac OS 9, displays a low-level system alert or dialog. The appearance and semantics of the alert or dialog can vary, depending on the error code passed in.

Not recommended

```
void SysError (
   short errorCode
);
```

Parameters

errorCode

A value or code that indicates a specific condition.

Discussion

A Carbon application running in Mac OS 9 can use this function to simulate a low-level system error. To exit in a more graceful manner when a serious error occurs, your application should call `ExitToShell` instead.

In Mac OS X, this function:

1. Prints a message containing the value of `errorCode` in the console log. Does not display a system alert or dialog.
2. If the variable `USERBREAK` is defined in the process environment, raises a `SIGINT` signal. Otherwise, returns control to the caller.

If a signal is raised and the caller does not handle the signal, the calling process is automatically terminated.

Availability
Not recommended in Carbon. Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later. Available in Mac OS X 10.0 and later.

Declared In
`MacErrors.h`

Data Types

OSErr

A numeric code used in Carbon to indicate the return status of a function.

```
typedef SInt16 OSErr;
```

Discussion
The system software sometimes uses error codes to inform an application that a requested service is not possible. Many functions return a result code of type `OSErr` that indicates whether the function completed successfully, and if not, what the reason for failure was.

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

Declared In
`IOMacOSTypes.h`

OSStatus

A numeric code used in Carbon to indicate the return status of a function.

```
typedef SInt32 OSStatus;
```

Discussion
The system software sometimes uses error codes to inform an application that a requested service is not possible. Many functions return a result code of type `OSStatus` that indicates whether the function completed successfully, and if not, what the reason for failure was.

If you want to use `OSStatus` to define error codes for your application, Apple recommends that you use values in the range 1000 through 9999 inclusive. Values outside of this range are reserved by Apple for internal use.

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

Declared In
`OSTypes.h`

Finder Interface Reference

Framework:	CoreServices/CoreServices.h, Carbon/Carbon.h
Declared in	Finder.h FinderRegistry.h

Overview

The Finder is an application that works with the system software to keep track of files and manage the user's desktop display. This document describes the programming interface your application should use to interact with the Finder. For example, you can use this interface to:

- Set up the resources the Finder needs to display and start up your application
- Set up the resources the Finder uses to display information about other files related to your application
- Examine or change Finder-related information stored in a volume's catalog file
- Support stationery pads

Data Types

CustomBadgeResource

```
struct CustomBadgeResource {
    SInt16 version;
    SInt16 customBadgeResourceID;
    OSType customBadgeType;
    OSType customBadgeCreator;
    OSType windowBadgeType;
    OSType windowBadgeCreator;
    OSType overrideType;
    OSType overrideCreator;
};
typedef struct CustomBadgeResource CustomBadgeResource;
typedef CustomBadgeResource * CustomBadgeResourcePtr;
```

Fields

`version`

`customBadgeResourceID`

`customBadgeType`

`customBadgeCreator`

`windowBadgeType`

`windowBadgeCreator`

`overrideType`

`overrideCreator`

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

Declared In
`Finder.h`

ExtendedFolderInfo

Defines an extended directory information structure. The `ExtendedFolderInfo` structure is preferred over the `DXInfo` structure.

```
struct ExtendedFolderInfo {
    Point scrollPosition;
    SInt32 reserved1;
    UInt16 extendedFinderFlags;
    SInt16 reserved2;
    SInt32 putAwayFolderID;
};
typedef struct ExtendedFolderInfo ExtendedFolderInfo;
```

Fields

`scrollPosition`

Scroll position within the Finder window. The Finder does not necessarily save this position immediately upon user action.

`reserved1`
Reserved (set to 0).

`extendedFinderFlags`
Extended Finder flags. See "Extended Finder Flags " (page 2262).

`reserved2`
Reserved (set to 0).

`putAwayFolderID`
If the user moves the folder onto the desktop, the directory ID of the folder from which the user moves it.

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

Declared In
`Finder.h`

FolderInfo

Defines a directory information structure. The `FolderInfo` structure is preferred over the `DInfo` structure.

```
struct FolderInfo {
    Rect windowBounds;
    UInt16 finderFlags;
    Point location;
    UInt16 reservedField;
};
typedef struct FolderInfo FolderInfo;
```

Fields
`windowBounds`
The rectangle for the window that the Finder displays when the user opens the folder.

`finderFlags`
Finder flags. See "Finder Flags" (page 2261).

`location`
Location of the folder in the parent window.

`reservedField`
Reserved. Set to 0.

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

Declared In
`Finder.h`

DInfo

Defines a directory information structure.

```
struct DInfo {
    Rect frRect;
    UInt16 frFlags;
    Point frLocation;
    SInt16 frView;
};
typedef struct DInfo DInfo;
```

Fields

`frRect`

The rectangle for the window that the Finder displays when the user opens the folder.

`frFlags`

Reserved.

`frLocation`

Location of the folder in the parent window.

`frView`

The manner in which folders are displayed; this is set by the user with commands from the View menu of the Finder.

Discussion

The Finder manipulates the fields in the directory information record. Your application shouldn't have to check or set any of these fields directly

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Finder.h`

DXInfo

Defines an extended directory information structure.

```
struct DXInfo {
    Point frScroll;
    SInt32 frOpenChain;
    SInt8 frScript;
    SInt8 frXFlags;
    SInt16 frComment;
    SInt32 frPutAway;
};
typedef struct DXInfo DXInfo;
```

Fields

`frScroll`

Scroll position within the Finder window. The Finder does not necessarily save this position immediately upon user action.

`frOpenChain`

Chain of directory IDs for open folders. The Finder numbers directory IDs. The Finder does not necessarily save this information immediately upon user action.

`frScript`
: Extended flags. If the high-bit is set, the script system for displaying the folder's name. Ordinarily, the Finder (and the Standard File Package) displays the names of all desktop objects in the current system script, which depends on the region-specific configuration of the system. The high bit of the byte in the fdScript field is set by default to 0, which causes the Finder to display the folder's name in the current system script. If the high bit is set to 1, the Finder (and the Standard File Package) displays the filename and directory name in the script whose code is recorded in the remaining 7 bits. However, as of system software version 7.1, the Window Manager and Dialog Manager do not support multiple simultaneous scripts, so the system script is always used for displaying filenames and directory names in dialog boxes, window titles, and other user interface elements used by the Finder. Therefore, until the system software's script capability is fully implemented, you should treat this field as reserved.

`frXFlags`
: Extended flags. See "Extended Finder Flags " (page 2262).

`frComment`
: Reserved (set to 0). If the high-bit is clear, an ID number for the comment that is displayed in the information window when the user selects a folder and chooses the Get Info command from the File menu. The numbers that identify comments are assigned by the Finder.

`frPutAway`
: If the user moves the folder onto the desktop, the directory ID of the folder from which the user moves it.

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

Declared In
`Finder.h`

ExtendedFileInfo

Defines an extended file information structure. The `ExtendedFileInfo` structure is preferred over the `FXInfo` structure.

```
struct ExtendedFileInfo {
    SInt16 reserved1[4];
    UInt16 extendedFinderFlags;
    SInt16 reserved2;
    SInt32 putAwayFolderID;
};
typedef struct ExtendedFileInfo ExtendedFileInfo;
```

Fields

`reserved1`
: Reserved (set to 0).

`extendedFinderFlags`
: Extended flags. See "Extended Finder Flags " (page 2262).

`reserved2`
: Reserved (set to 0).

`putAwayFolderID`
: If the user moves the file onto the desktop, the directory ID of the folder from which the user moves the file.

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

Declared In
`Finder.h`

FileInfo

Defines a file information structure. The `FileInfo` structure is preferred over the `FInfo` structure.

```
struct FileInfo {
    OSType fileType;
    OSType fileCreator;
    UInt16 finderFlags;
    Point location;
    UInt16 reservedField;
};
typedef struct FileInfo FileInfo;
```

Fields

`fileType`
File type.

`fileCreator`
The signature of the application that created the file.

`finderFlags`
Finder flags. See "Finder Flags" (page 2261).

`location`
The location--specified in coordinates local to the window--of the file's icon within its window.

`reservedField`
The window in which the file's icon appears; this information is meaningful only to the Finder.

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

Declared In
`Finder.h`

FInfo

Defines a file information structure.

```
struct FInfo {
    OSType fdType;
    OSType fdCreator;
    UInt16 fdFlags;
    Point fdLocation;
    SInt16 fdFldr;
};
typedef struct FInfo FInfo;
```

Fields

`fdType`
File type.

`fdCreator`
The signature of the application that created the file.

`fdFlags`
Finder flags. See "Finder Flags" (page 2261).

`fdLocation`
The location--specified in coordinates local to the window--of the file's icon within its window.

`fdFldr`
The window in which the file's icon appears; this information is meaningful only to the Finder.

Discussion

You typically set a file's type and creator when you create the file. The Finder manipulates the other fields in the file information record, which is a data structure of type `FInfo`. After you have created a file, you can use the File Manager function `FSpGetFInfo` to return the file information record, then change the `fdType` and `fdCreator` fields by using the File Manager function `FSpSetFInfo`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Finder.h`

FXInfo

Defines an extended file information structure.

```
struct FXInfo {
    SInt16 fdIconID;
    SInt16 fdReserved[3];
    SInt8 fdScript;
    SInt8 fdXFlags;
    SInt16 fdComment;
    SInt32 fdPutAway;
};
typedef struct FXInfo FXInfo;
```

Fields

`fdIconID`
An ID number for the file's icon; the numbers that identify icons are assigned by the Finder.

`fdReserved`
Reserved.

`fdScript`
: Extended flags. Script code if high-bit is set.

`fdXFlags`
: Extended flags.

`fdComment`
: Reserved (set to 0). If the high-bit is clear, an ID number for the comment that is displayed in the information window when the user selects a file and chooses the Get Info command from the File menu. The numbers that identify comments are assigned by the Finder.

`fdPutAway`
: If the user moves the file onto the desktop, the directory ID of the folder from which the user moves the file.

Discussion
The Finder manipulates the fields in the extended file information records ; your application shouldn't have to check or set any of these fields directly.

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

Declared In
`Finder.h`

RoutingResourceEntry

```
struct RoutingResourceEntry {
    OSType creator;
    OSType fileType;
    OSType targetFolder;
    OSType destinationFolder;
    OSType reservedField;
};
typedef struct RoutingResourceEntry RoutingResourceEntry;
typedef RoutingResourceEntry * RoutingResourcePtr;
```

Fields
`creator`
`fileType`
`targetFolder`
`destinationFolder`
`reservedField`

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

Declared In
`Finder.h`

Constants

Finder Flags

Specify flags used by the Finder.

```
enum {
    kIsOnDesk = 0x0001,
    kColor = 0x000E,
    kIsShared = 0x0040,
    kHasNoINITs = 0x0080,
    kHasBeenInited = 0x0100,
    kHasCustomIcon = 0x0400,
    kIsStationery = 0x0800,
    kNameLocked = 0x1000,
    kHasBundle = 0x2000,
    kIsInvisible = 0x4000,
    kIsAlias = 0x8000
};
```

Constants

`kIsOnDesk`

Unused and reserved in System 7; set to 0.

Available in Mac OS X v10.0 and later.

Declared in `Finder.h`.

`kColor`

Three bits of color coding.

Available in Mac OS X v10.0 and later.

Declared in `Finder.h`.

`kIsShared`

The file is an application that can be executed by multiple users simultaneously. Defined only for applications; otherwise, set to 0.

Available in Mac OS X v10.0 and later.

Declared in `Finder.h`.

`kHasNoINITs`

The file contains no 'INIT' resources; set to 0. Reserved for directories; set to 0.

Available in Mac OS X v10.0 and later.

Declared in `Finder.h`.

`kHasBeenInited`

The Finder has recorded information from the file's bundle resource into the desktop database and given the file or folder a position on the desktop.

Available in Mac OS X v10.0 and later.

Declared in `Finder.h`.

`kHasCustomIcon`

The file or directory contains a customized icon.

Available in Mac OS X v10.0 and later.

Declared in `Finder.h`.

`kIsStationery`
For a file, this bit indicates that the file is a stationery pad. For directories, this bit is reserved--in which case, set to 0.

Available in Mac OS X v10.0 and later.

Declared in `Finder.h`.

`kNameLocked`
The file or directory can't be renamed from the Finder, and the icon cannot be changed.

Available in Mac OS X v10.0 and later.

Declared in `Finder.h`.

`kHasBundle`
For a file, this bit indicates that the file contains a bundle resource. For a directory, this bit indicates that the directory is a file package. Note that not all file packages have this bit set; many file packages are identified by other means, such as a recognized package extension in the name. The proper way to determine if an item is a package is through Launch Services.

Available in Mac OS X v10.0 and later.

Declared in `Finder.h`.

`kIsInvisible`
The file or directory is invisible from the Finder and from the Navigation Services dialogs.

Available in Mac OS X v10.0 and later.

Declared in `Finder.h`.

`kIsAlias`
For a file, this bit indicates that the file is an alias file. For directories, this bit is reserved--in which case, set to 0.

Available in Mac OS X v10.0 and later.

Declared in `Finder.h`.

Extended Finder Flags

Define flags used by the extended file and folder information structures.

```
enum {
    kExtendedFlagsAreInvalid = 0x8000,
    kExtendedFlagHasCustomBadge = 0x0100,
    kExtendedFlagHasRoutingInfo = 0x0004
};
```

Constants

`kExtendedFlagsAreInvalid`
If set the other extended flags are ignored.

Available in Mac OS X v10.0 and later.

Declared in `Finder.h`.

`kExtendedFlagHasCustomBadge`
Set if the file or folder has a badge resource.

Available in Mac OS X v10.0 and later.

Declared in `Finder.h`.

`kExtendedFlagHasRoutingInfo`

Set if the file contains routing info resource.

Available in Mac OS X v10.0 and later.

Declared in `Finder.h`.

Finder Error Codes

```
enum {
    errFinderIsBusy = -15260,
    errFinderWindowNotOpen = -15261,
    errFinderCannotPutAway = -15262,
    errFinderWindowMustBeIconView = -15263,
    errFinderWindowMustBeListView = -15264,
    errFinderCantMoveToDestination = -15265,
    errFinderCantMoveSource = -15266,
    errFinderCantOverwrite = -15267,
    errFinderIncestuousMove = -15268,
    errFinderCantMoveToAncestor = -15269,
    errFinderCantUseTrashedItems = -15270,
    errFinderItemAlreadyInDest = -15271,
    errFinderUnknownUser = -15272,
    errFinderSharePointsCantInherit = -15273,
    errFinderWindowWrongType = -15274,
    errFinderPropertyNowWindowBased = -15275,
    errFinderAppFolderProtected = -15276,
    errFinderSysFolderProtected = -15277,
    errFinderBoundsWrong = -15278,
    errAEValueOutOfRange = -15279,
    errFinderPropertyDoesNotApply = -15280,
    errFinderFileSharingMustBeOn = -15281,
    errFinderMustBeActive = -15282,
    errFinderVolumeNotFound = -15283,
    errFinderLockedItemsInTrash = -15284,
    errFinderOnlyLockedItemsInTrash = -15285,
    errFinderProgramLinkingMustBeOn = -15286,
    errFinderWindowMustBeButtonView = -15287,
    errFinderBadPackageContents = -15288,
    errFinderUnsupportedInsidePackages = -15289,
    errFinderCorruptOpenFolderList = -15290,
    errFinderNoInvisibleFiles = -15291,
    errFinderCantDeleteImmediately = -15292,
    errFinderLastReserved = -15379
};
```

Finder Events

```
enum {
    kAECleanUp = 'fclu',
    kAEEject = 'ejct',
    kAEEmpty = 'empt',
    kAEErase = 'fera',
    kAEGestalt = 'gstl',
    kAEPutAway = 'ptwy',
    kAERebuildDesktopDB = 'rddb',
    kAESync = 'fupd',
    kAEInterceptOpen = 'fopn'
};
```

kAEDatabaseSuite

```
enum {
    kAEDatabaseSuite = 'DATA',
    kAESort = 'SORT'
};
```

kAEFinderSuite

```
enum {
    kAEFinderSuite = 'fndr'
};
```

cAliasFile

```
enum {
    cAliasFile = 'alia',
    cApplicationFile = 'appf',
    cControlPanelFile = 'ccdv',
    cDeskAccessoryFile = 'dafi',
    cDocumentFile = 'docf',
    cFontFile = 'fntf',
    cSoundFile = 'sndf',
    cClippingFile = 'clpf',
    cContainer = 'ctnr',
    cDesktop = 'cdsk',
    cSharableContainer = 'sctr',
    cDisk = 'cdis',
    cFolder = 'cfol',
    cSuitcase = 'stcs',
    cAccessorySuitcase = 'dsut',
    cFontSuitcase = 'fsut',
    cTrash = 'ctrs',
    cDesktopPrinter = 'dskp',
    cPackage = 'pack',
    cContentSpace = 'dwnd',
    cContainerWindow = 'cwnd',
    cInfoWindow = 'iwnd',
    cSharingWindow = 'swnd',
    cStatusWindow = 'qwnd',
    cClippingWindow = 'lwnd',
    cPreferencesWindow = 'pwnd',
    cDTPWindow = 'dtpw',
    cProcess = 'prcs',
    cAccessoryProcess = 'pcda',
    cApplicationProcess = 'pcap',
    cGroup = 'sgrp',
    cUser = 'cuse',
    cSharingPrivileges = 'priv',
    cPreferences = 'cprf',
    cLabel = 'clbl',
    cSound = 'snd ',
    cAliasList = 'alst',
    cSpecialFolders = 'spfl',
    cOnlineDisk = 'cods',
    cOnlineLocalDisk = 'clds',
```

```
    cOnlineRemoteDisk = 'crds',
    cEntireContents = 'ects',
    cIconFamily = 'ifam'
};
```

cInternalFinderObject

```
enum {
    cInternalFinderObject = 'obj '
};
```

enumAllDocuments

```
enum {
    enumAllDocuments = 'alld',
    enumFolders = 'fold',
    enumAliases = 'alia',
    enumStationery = 'stat'
};
```

enumArrangement

```
enum {
    enumArrangement = 'earr'
};
```

enumDate

```
enum {
    enumDate = 'enda',
    enumAnyDate = 'anyd',
    enumToday = 'tday',
    enumYesterday = 'yday',
    enumThisWeek = 'twek',
    enumLastWeek = 'lwek',
    enumThisMonth = 'tmon',
    enumLastMonth = 'lmon',
    enumThisYear = 'tyer',
    enumLastYear = 'lyer',
    enumBeforeDate = 'bfdt',
    enumAfterDate = 'afdt',
    enumBetweenDate = 'btdt',
    enumOnDate = 'ondt'
};
```

enumIconSize

```
enum {
```

```
    enumIconSize = 'isiz',
    enumSmallIconSize = pSmallIcon,
    enumMiniIconSize = 'miic',
    enumLargeIconSize = 'lgic'
};
```

enumInfoWindowPanel

```
enum {
    enumInfoWindowPanel = 'ipnl',
    enumGeneralPanel = 'gpnl',
    enumSharingPanel = 'spnl',
    enumStatusNConfigPanel = 'scnl',
    enumFontsPanel = 'fpnl',
    enumMemoryPanel = 'mpnl'
};
```

enumPrefsWindowPanel

```
enum {
    enumPrefsWindowPanel = 'pple',
    enumPrefsGeneralPanel = 'pgnp',
    enumPrefsLabelPanel = 'plbp',
    enumPrefsIconViewPanel = 'pivp',
    enumPrefsButtonViewPanel = 'pbvp',
    enumPrefsListViewPanel = 'plvp'
};
```

enumSortDirection

```
enum {
    enumSortDirection = 'sodr',
    enumSortDirectionNormal = 'snrm',
    enumSortDirectionReverse = 'srvs'
};
```

enumViewBy

```
enum {
    enumViewBy = 'vwby',
    enumGestalt = 'gsen',
    enumConflicts = 'cflc',
    enumExistingItems = 'exsi',
    enumOlderItems = 'oldr'
};
```

enumWhere

```
enum {
    enumWhere = 'wher',
    enumAllLocalDisks = 'aldk',
    enumAllRemoteDisks = 'ardk',
    enumAllDisks = 'alld',
    enumAllOpenFolders = 'aofo'
};
```

fOnDesk

Obsolete. Use the constants described in "Finder Flags."

```
enum {
    fOnDesk = kIsOnDesk,
    fHasBundle = kHasBundle,
    fInvisible = kIsInvisible
};
```

formAlias

```
enum {
    formAlias = typeAlias,
    formCreator = pFileCreator
};
```

fTrash

Obsolete.

```
enum {
    fTrash = -3,
    fDesktop = -2,
    fDisk = 0
};
```

Clipping File Creator and Types

```
enum {
    kClippingCreator = 'drag',
    kClippingPictureType = 'clpp',
    kClippingTextType = 'clpt',
    kClippingSoundType = 'clps',
    kClippingUnknownType = 'clpu'
};
```

kContainerFolderAliasType

```
enum {
    kContainerFolderAliasType = 'fdrp',
    kContainerTrashAliasType = 'trsh',
    kContainerHardDiskAliasType = 'hdsk',
    kContainerFloppyAliasType = 'flpy',
    kContainerServerAliasType = 'srvr',
    kApplicationAliasType = 'adrp',
    kContainerAliasType = 'drop',
    kDesktopPrinterAliasType = 'dtpa',
    kContainerCDROMAliasType = 'cddr',
    kApplicationCPAliasType = 'acdp',
    kApplicationDAAliasType = 'addp',
    kPackageAliasType = 'fpka',
    kAppPackageAliasType = 'fapa'
};
```

kCustomBadgeResourceType

```
enum {
    kCustomBadgeResourceType = 'badg',
    kCustomBadgeResourceID = kCustomIconResource,
    kCustomBadgeResourceVersion = 0
};
```

kCustomIconResource

```
enum {
    kCustomIconResource = -16455
};
```

kExportedFolderAliasType

```
enum {
    kExportedFolderAliasType = 'faet',
    kDropFolderAliasType = 'fadr',
    kSharedFolderAliasType = 'fash',
    kMountedFolderAliasType = 'famn'
};
```

keyASPrepositionHas

```
enum {
    keyASPrepositionHas = 'has ',
    keyAll = 'kyal',
    keyOldFinderItems = 'fsel'
};
```

Constants

`keyASPrepositionHas`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`keyAll`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`keyOldFinderItems`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

keyIconAndMask

```
enum {
    keyIconAndMask = 'ICN#',
    key32BitIcon = 'il32',
    key8BitIcon = 'icl8',
    key4BitIcon = 'icl4',
    key8BitMask = 'l8mk',
    keySmallIconAndMask = 'ics#',
    keySmall8BitIcon = 'ics8',
    keySmall4BitIcon = 'ics4',
    keySmall32BitIcon = 'is32',
    keySmall8BitMask = 's8mk',
    keyMini1BitMask = 'icm#',
    keyMini4BitIcon = 'icm4',
    keyMini8BitIcon = 'icm8',
    keyAEUsing = 'usin',
    keyAEReplacing = 'alrp',
    keyAENoAutoRouting = 'rout',
    keyLocalPositionList = 'mvpl',
    keyGlobalPositionList = 'mvpg',
    keyRedirectedDocumentList = 'fpdl'
};
```

Constants

`keyIconAndMask`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`key32BitIcon`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`key8BitIcon`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`key4BitIcon`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`key8BitMask`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`keySmallIconAndMask`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`keySmall8BitIcon`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`keySmall4BitIcon`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`keySmall32BitIcon`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`keySmall8BitMask`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`keyMini1BitMask`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`keyMini4BitIcon`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`keyMini8BitIcon`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`keyAEUsing`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`keyAEReplacing`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`keyAENoAutoRouting`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`keyLocalPositionList`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`keyGlobalPositionList`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

`keyRedirectedDocumentList`
Available in Mac OS X v10.0 and later.
Declared in `FinderRegistry.h`.

kFirstMagicBusyFiletype

```
enum {
    kFirstMagicBusyFiletype = 'bzy ',
    kLastMagicBusyFiletype = 'bzy?'
};
```

kInternetLocationCreator

```
enum {
    kInternetLocationCreator = 'drag',
    kInternetLocationHTTP = 'ilht',
    kInternetLocationFTP = 'ilft',
    kInternetLocationFile = 'ilfi',
    kInternetLocationMail = 'ilma',
    kInternetLocationNNTP = 'ilnw',
    kInternetLocationAFP = 'ilaf',
    kInternetLocationAppleTalk = 'ilat',
    kInternetLocationNSL = 'ilns',
    kInternetLocationGeneric = 'ilge'
};
```

kIsStationary

```
enum {
    kIsStationary = kIsStationery
};
```

kMagicBusyCreationDate

```
enum {
    kMagicBusyCreationDate = 0x4F3AFDB0
};
```

kRoutingResourceType

```
enum {
    kRoutingResourceType = 'rout',
    kRoutingResourceID = 0
};
```

kSystemFolderAliasType

```
enum {
    kSystemFolderAliasType = 'fasy',
    kAppleMenuFolderAliasType = 'faam',
    kStartupFolderAliasType = 'fast',
```

```
    kPrintMonitorDocsFolderAliasType = 'fapn',
    kPreferencesFolderAliasType = 'fapf',
    kControlPanelFolderAliasType = 'fact',
    kExtensionFolderAliasType = 'faex'
};
```

pAboutMacintosh

```
enum {
    pAboutMacintosh = 'abbx',
    pAppleMenuItemsFolder = 'amnu',
    pControlPanelsFolder = 'ctrl',
    pDesktop = 'desk',
    pExtensionsFolder = 'extn',
    pFinderPreferences = 'pfrp',
    pFontsFolder = 'font',
    pFontsFolderPreAllegro = 'ffnt',
    pLargestFreeBlock = 'mfre',
    pPreferencesFolder = 'pref',
    pShortCuts = 'scut',
    pShutdownFolder = 'shdf',
    pStartupItemsFolder = 'strt',
    pSystemFolder = 'macs',
    pTemporaryFolder = 'temp',
    pViewPreferences = 'pvwp',
    pStartingUp = 'awak'
};
```

pApplicationFile

```
enum {
    pApplicationFile = cApplicationFile
};
```

pCanConnect

```
enum {
    pCanConnect = 'ccon',
    pCanChangePassword = 'ccpw',
    pCanDoProgramLinking = 'ciac',
    pIsOwner = 'isow',
    pARADialIn = 'arad',
    pShouldCallBack = 'calb',
    pCallBackNumber = 'cbnm'
};
```

pCapacity

```
enum {
    pCapacity = 'capa',
```

```
    pEjectable = 'isej',
    pFreeSpace = 'frsp',
    pLocal = 'isrv',
    pIsStartup = 'istd'
};
```

pComment

```
enum {
    pComment = 'comt',
    pContainer = cContainer,
    pContentSpace = cContentSpace,
    pCreationDateOld = 'crtd',
    pCreationDate = 'ascd',
    pDescription = 'dscr',
    pDisk = cDisk,
    pFolderOld = cFolder,
    pFolder = 'asdr',
    pIconBitmap = 'iimg',
    pInfoWindow = cInfoWindow,
    pKind = 'kind',
    pLabelIndex = 'labi',
    pModificationDateOld = 'modd',
    pModificationDate = 'asmo',
    pPhysicalSize = 'phys',
    pPosition = 'posn',
    pIsSelected = 'issl',
    pSize = pPointSize,
    pWindow = cWindow,
    pPreferencesWindow = cPreferencesWindow
};
```

pCompletelyExpanded

```
enum {
    pCompletelyExpanded = 'pexc',
    pContainerWindow = cContainerWindow,
    pEntireContents = cEntireContents,
    pExpandable = 'pexa',
    pExpanded = 'pexp',
    pPreviousView = 'svew',
    pView = 'pvew',
    pIconSize = pListViewIconSize,
    pKeepArranged = 'arrg',
    pKeepArrangedBy = 'arby'
};
```

pDeskAccessoryFile

```
enum {
    pDeskAccessoryFile = cDeskAccessoryFile
};
```

pFile

```
enum {
    pFile = cFile,
    pPartitionSpaceUsed = 'pusd',
    pLocalAndRemoteEvents = 'revt',
    pHasScriptingTerminology = 'hscr'
};
```

pFileCreator

```
enum {
    pFileCreator = 'fcrt',
    pFileType = 'asty',
    pFileTypeOld = 'fitp',
    pIsLocked = 'aslk',
    pIsLockedOld = 'islk',
    pProductVersion = 'ver2'
};
```

pFileShareOn

```
enum {
    pFileShareOn = 'fshr',
    pFileShareStartingUp = 'fsup',
    pProgramLinkingOn = 'iac '
};
```

pInfoPanel

```
enum {
    pInfoPanel = 'panl'
};
```

pInternetLocation

```
enum {
    pInternetLocation = 'iloc'
};
```

pIsZoomedFull

```
enum {
    pIsZoomedFull = 'zumf',
    pIsPopup = 'drwr',
    pIsPulledOpen = 'pull',
    pIsCollapsed = 'wshd'
```

```
};
```

pMinAppPartition

```
enum {
    pMinAppPartition = 'mprt',
    pAppPartition = 'appt',
    pSuggestedAppPartition = 'sprt',
    pIsScriptable = 'isab'
};
```

pNoArrangement

```
enum {
    pNoArrangement = 'narr',
    pSnapToGridArrangement = 'grda',
    pByNameArrangement = 'nama',
    pByModificationDateArrangement = 'mdta',
    pByCreationDateArrangement = 'cdta',
    pBySizeArrangement = 'siza',
    pByKindArrangement = 'kina',
    pByLabelArrangement = 'laba'
};
```

pObject

```
enum {
    pObject = cObject
};
```

pOriginalItem

```
enum {
    pOriginalItem = 'orig'
};
```

pOwner

```
enum {
    pOwner = 'sown',
    pOwnerPrivileges = 'ownr',
    pGroup = cGroup,
    pGroupPrivileges = 'gppr',
    pGuestPrivileges = 'gstp',
    pArePrivilegesInherited = 'iprv',
    pExported = 'sexp',
    pMounted = 'smou',
    pSharingProtection = 'spro',
```

```
    pSharing = 'shar',
    pSharingWindow = cSharingWindow
};
```

pSeeFiles

```
enum {
    pSeeFiles = 'prvr',
    pSeeFolders = 'prvs',
    pMakeChanges = 'prvw'
};
```

pSharableContainer

```
enum {
    pSharableContainer = cSharableContainer
};
```

pShowFolderSize

```
enum {
    pShowFolderSize = 'sfsz',
    pShowComment = 'scom',
    pShowDate = 'sdat',
    pShowCreationDate = 'scda',
    pShowKind = 'sknd',
    pShowLabel = 'slbl',
    pShowSize = 'ssiz',
    pShowVersion = 'svrs',
    pSortDirection = 'sord',
    pShowDiskInfo = 'sdin',
    pListViewIconSize = 'lvis',
    pGridIcons = 'fgrd',
    pStaggerIcons = 'fstg',
    pViewFont = 'vfnt',
    pViewFontSize = 'vfsz'
};
```

pShowModificationDate

```
enum {
    pShowModificationDate = pShowDate,
    pUseRelativeDate = 'urdt',
    pDelayBeforeSpringing = 'dela',
    pSpringOpenFolders = 'sprg',
    pUseShortMenus = 'usme',
    pUseWideGrid = 'uswg',
    pLabel1 = 'lbl1',
    pLabel2 = 'lbl2',
    pLabel3 = 'lbl3',
```

```
    pLabel4 = 'lbl4',
    pLabel5 = 'lbl5',
    pLabel6 = 'lbl6',
    pLabel7 = 'lbl7',
    pDefaultIconViewIconSize = 'iisz',
    pDefaultButtonViewIconSize = 'bisz',
    pDefaultListViewIconSize = 'lisz',
    pIconViewArrangement = 'iarr',
    pButtonViewArrangement = 'barr'
};
```

pSmallIcon

```
enum {
    pSmallIcon = 'smic',
    pSmallButton = 'smbu',
    pLargeButton = 'lgbu',
    pGrid = 'grid'
};
```

pSound

```
enum {
    pSound = 'snd '
};
```

pStartupDisk

```
enum {
    pStartupDisk = 'sdsk',
    pTrash = 'trsh'
};
```

pWarnOnEmpty

```
enum {
    pWarnOnEmpty = 'warn'
};
```

typeIconFamily

```
enum {
    typeIconFamily = cIconFamily,
    typeIconAndMask = 'ICN#',
    type8BitMask = 'l8mk',
    type32BitIcon = 'il32',
    type8BitIcon = 'icl8',
    type4BitIcon = 'icl4',
```

```
    typeSmallIconAndMask = 'ics#',
    typeSmall8BitMask = 's8mk',
    typeSmall32BitIcon = 'is32',
    typeSmall8BitIcon = 'ics8',
    typeSmall4BitIcon = 'ics4',
    typeRelativeTime = 'rtim',
    typeConceptualTime = 'timc'
};
```

Result Codes

The most common result codes returned by Finder Interface are listed below.

Result Code	Value	Description
errOffsetInvalid	-1800	Available in Mac OS X v10.0 and later.
errOffsetIsOutsideOfView	-1801	Available in Mac OS X v10.0 and later.
errTopOfDocument	-1810	Available in Mac OS X v10.0 and later.
errTopOfBody	-1811	Available in Mac OS X v10.0 and later.
errEndOfDocument	-1812	Available in Mac OS X v10.0 and later.
errEndOfBody	-1813	Available in Mac OS X v10.0 and later.
desktopDamagedErr	-1305	The desktop database has become corrupted Available in Mac OS X v10.0 and later.

MDImporter Reference

Framework:	CoreServices/CoreServices.h
Declared in	MDImporter.h
Companion guides	Spotlight Overview Spotlight Importer Programming Guide Plug-ins

Overview

An MDImporter is responsible for returning the metadata contained with a file. The Spotlight server is notified when a file is changed and loads the appropriate importer to extract the metadata. An importer is implemented as a CFPlugin.

Callbacks

ImporterImportData

Defines a pointer to an importer import callback that imports importers.

```
typedef Boolean (* ImporterImportData)
(
void *thisInterface,
CFMutableDictionaryRef attributes,
CFStringRef contentTypeUTI,
CFStringRef pathToFile
)
```

If you name your function `GetMetadataForFile`, you would declare it like this:

```
Boolean GetMetadataForFile
(
void *thisInterface,
CFMutableDictionaryRef attributes,
CFStringRef contentTypeUTI,
CFStringRef pathToFile
)
```

Parameters

thisInterface

The CFPlugin object that is called. This value is passed to the callback function.

attributes

A mutable dictionary that you should populate with the metadata key/value pairs. This dictionary is created and passed to the callback function.

contentTypeUTI

The content type of the file as a uniform type identifier. This value is passed to the callback function.

pathToFile

The full path to the file. This value is passed to the callback function.

Return Value

Your callback function should return `true` if the metadata was successfully returned, `false` if the metadata was not returned.

Constants

kMDImporterTypeID

Type ID of an importer plug-in.

```
#define kMDImporterTypeID
CFUUIDGetConstantUUIDWithBytes(kCFAllocatorDefault,0x8B,0x08,0xC4,0xBF,0x41,0x5
B,0x11,0xD8,0xB3,0xF9,0x00,0x03,0x93,0x67,0x26,0xFC);
```

Constants

`kMDImporterTypeID`

Only importers with this type ID are loaded.

Available in Mac OS X v10.4 and later.

Declared in `MDImporter.h`.

Discussion

The string representation of this UUID is `8B08C4BF-415B-11D8-B3F9-0003936726FC`.

kMDImporterInterfaceID

Interface required by a importer plug-in.

```
#define kMDImporterInterfaceID
CFUUIDGetConstantUUIDWithBytes(kCFAllocatorDefault,0x6E,0xBC,0x27,0xC4,0x89,0x9
C,0x11,0xD8,0x84,0xAE,0x00,0x03,0x93,0x67,0x26,0xFC);
```

Constants

`kMDImporterInterfaceID`

Importers must implement an interface corresponding to this UUID.

Available in Mac OS X v10.4 and later.

Declared in `MDImporter.h`.

Discussion

The string representation of this UUID is `6EBC27C4-899C-11D8-84A3-0003936726FC`.

MDSchema Reference

Framework:	CoreServices/CoreServices.h
Declared in	MDSchema.h
Companion guide	Spotlight Overview

Overview

The MDSchema functions provide information about the metadata returned for an item including the type of metadata provided for a file type, the localized display name for a metadata attribute key, and the schema for a metadata attribute key.

Functions

MDSchemaCopyAllAttributes

Returns an array containing all the metadata attributes defined in the schema.

```
CFArrayRef MDSchemaCopyAllAttributes (
   void
);
```

Availability
Available in Mac OS X version 10.4 or later.

Declared In
`MDSchema.h`

MDSchemaCopyAttributesForContentType

Returns a dictionary containing the metadata attributes for the specified UTI type.

```
MD_BEGIN_C_DECLS CFDictionaryRef MDSchemaCopyAttributesForContentType (
   CFStringRef contentTypeUTI
);
```

Parameters
utiType
The UTI type.

Return Value
A dictionary containing `kMDAttributeDisplayValues` and `kMDAttributeAllValues` keys. Returns `NULL` if the UTI type is unknown.

Discussion
This function returns the metadata attributes for the specified UTI type only.

Availability
Available in Mac OS X version 10.4 or later.

Declared In
`MDSchema.h`

MDSchemaCopyDisplayDescriptionForAttribute

Returns the localized description of a metadata attribute key.

```
CFStringRef MDSchemaCopyDisplayDescriptionForAttribute (
   CFStringRef name
);
```

Parameters

name
The name of the metadata attribute key.

Return Value
The localized description of the metadata attribute, or `NULL` if no localized description is available.

Availability
Available in Mac OS X version 10.4 or later.

Declared In
`MDSchema.h`

MDSchemaCopyDisplayNameForAttribute

Returns the localized display name of a metadata attribute key.

```
CFStringRef MDSchemaCopyDisplayNameForAttribute (
   CFStringRef name
);
```

Parameters

name
The name of the metadata attribute key.

Return Value
The localized display name of the metadata attribute, or `NULL` if no localized display name is available.

Availability
Available in Mac OS X version 10.4 or later.

Declared In
`MDSchema.h`

MDSchemaCopyMetaAttributesForAttribute

Returns a dictionary describing the values for the specified metadata attribute key.

```
CFDictionaryRef MDSchemaCopyMetaAttributesForAttribute (
   CFStringRef name
);
```

Parameters

name

The name of the metadata attribute key.

Return Value

A dictionary describing the schema of the metadata attribute key.

Availability

Available in Mac OS X version 10.4 or later.

Declared In

`MDSchema.h`

Constants

Available Metadata Attribute Keys

Specify the available metadata attribute keys for a content type.

```
const CFStringRef     kMDAttributeDisplayValues;
const CFStringRef     kMDAttributeAllValues;
```

Constants

`kMDAttributeDisplayValues`

An array of strings containing the available display metadata attribute keys, or `NULL` if the type is not known by the system.

Available in Mac OS X v10.4 and later.

Declared in `MDSchema.h`.

`kMDAttributeAllValues`

An array of strings containing available the metadata attribute keys, or `NULL` if the type is not known by the system.

Available in Mac OS X v10.4 and later.

Declared in `MDSchema.h`.

Discussion

These keys are in the dictionary returned by the function `MDSchemaCopyAttributesForContentType`.

Availability

Available in Mac OS X version 10.4 and later.

Metadata Attribute Schema Description Keys

Specify the schema of a metadata attribute key.

```
const CFStringRef kMDAttributeName;
const CFStringRef kMDAttributeType;
const CFStringRef kMDAttributeMultiValued;
```

Constants

`kMDAttributeName`

A string containing the name of the metadata attribute key.

Available in Mac OS X v10.4 and later.

Declared in `MDSchema.h`.

`kMDAttributeType`

A CFNumberRef or CFTypeId describing the type of data returned as the value of the metadata attribute key.

Available in Mac OS X v10.4 and later.

Declared in `MDSchema.h`.

`kMDAttributeMultiValued`

A boolean that indicates if the metadata attribute value is multi-valued. If this is `TRUE`, the metadata attribute value is an array of the types specified in `kMDAttributeType`.

Available in Mac OS X v10.4 and later.

Declared in `MDSchema.h`.

Discussion

These keys are in the dictionary returned by the function `MDSchemaCopyMetaAttributesForAttribute`.

Availability

Available in Mac OS X version 10.4 and later.

Open Transport Reference

Framework:	CoreServices/CoreServices.h
Declared in	OpenTransport.h OpenTransportProviders.h OpenTransportProtocol.h

Overview

Open Transport is the Mac OS 8 and 9 API for accessing TCP/IP networks, such as the Internet, at the transport level. For Mac OS X, Apple provides Open Transport as a compatibility library to ease migration of legacy applications. As such, Mac OS X does not support the entire Open Transport API.

In new Mac OS X applications you should not use Open Transport but should instead use BSD Sockets or, when possible, higher-level Core Services and Core Foundation APIs such as CFNetwork, CFURL, CFSocket, and CFStream. You can also use Cocoa networking classes such as NSURL, NSURLHandle, and NSNetService.

In Mac OS X, Open Transport provides limited support for endpoints and port access, and no support for the XTI or UNIX STREAMS interfaces. If you want your application to run in Mac OS 8 and 9 and in Mac OS X, use Open Transport for your Mac OS 8 and 9 version and Apple's newer APIs for your Mac OS X version.

For more information about Open Transport, see:

http://developer.apple.com/macos/opentransport/

Mac OS X supports only these Open Transport providers:

- TCP, UDP, and Raw IP Endpoints
- TCP/IP Services Providers and TCP/IP Mapper Providers (for the Domain Name Resolver protocol)
- DDP endpoints, AppleTalk Services Providers, and AppleTalk Mappers (for the Name Binding Protocol)
- OT/PPP endpoints

Mac OS X does not support ADSP, ATP, ASP, PAP, or serial endpoints.

You may have to revise your code if it uses Open Transport in one of the following ways:

- Your application uses a function that directly gains access to a network port. Ports are read-only in Mac OS X. Code that communicates directly with network interfaces must use the IOKit API.
- Your application uses the transaction-based endpoint feature of Open Transport. This feature is not supported in Mac OS X. Removal of this capability should affect only users of AppleTalk protocols such as ASP.

- Your application uses Open Transport's XTI interfaces or UNIX STREAMS interfaces. Mac OS X will not support these interfaces. You can obtain similar functionality using supported high-level functions.
- In Mac OS X, one cannot assume that Open Transport deferred tasks and notifiers procedures run at deferred task level. They may be preempted by the main event loop or another Mac OS X thread. You should always use atomic operations to access data shared between deferred tasks and notifiers and main system tasks.

Mac OS X does not support functions for:

- accessing Open Transport hash lists
- accessing the Open Transport port name or icon
- directly manipulating CFM or ASLM libraries

Client context parameters have been added to a number of OT functions. (An OT client is an application or a shared library.) Each client of Open Transport now has its own client context so that OT can track resources it allocates on behalf of the client. OT resources are objects like endpoints, timer tasks, and blocks of memory. To find out more about Open Transport resources management, see *"Understanding Open Transport Asset Tracking"* at:

http://developer.apple.com/technotes/tn/tn1173.html

Mac OS X introduces a new type, `OTClientContextPtr`, that represents the OT client context. This new type is passed as an extra parameter to functions that allocate OT resources. Before Mac OS X, the OT client context was determined by the Open Transport static libraries that you linked to your application. Now the OT client context is determined explicitly. The same Carbon binary can run on Mac OS 8/9 and Mac OS X, and you do not have to link your application to the static libraries.

You can use `InitOpenTransportinContext` to replace `InitOpenTransport`. It functions identically except that it also takes a client context pointer and a flags parameter to indicate whether you are initializing OT for an application or a shared library. When your application or shared library is done using Open Transport you should call `CloseOpenTransportInContext` to dispose of the Open Transport resources allocated for the client.

The following functions now take a client context:

- `CloseOpenTransportInContext` (page 2301)
- `OTAllocInContext` (page 2308)
- `OTAllocMemInContext` (page 2309)
- `OTAsyncOpenAppleTalkServicesInContext` (page 2310)
- `OTAsyncOpenEndpointInContext` (page 2310)
- `OTAsyncOpenInternetServicesInContext` (page 2311)
- `OTAsyncOpenMapperInContext` (page 2312)
- `OTCreateDeferredTaskInContext` (page 2329)
- `OTCreateTimerTaskInContext` (page 2330)
- `OTOpenAppleTalkServicesInContext` (page 2372)
- `OTOpenEndpointInContext` (page 2373)

- OTOpenInternetServicesInContext (page 2374)
- OTOpenMapperInContext (page 2374)

As a convenience, applications may pass a null pointer to these routines and Open Transport will use the context that was passed to InitOpenTransport. However, shared libraries must always pass a valid OTClientContextPtr.

If you want to keep your application source code compatible with pre-Mac OS X systems, you may define the C preprocessor constant OTCARBONAPPLICATION to 1 to use the old routine names without the "InContext" suffix.

Mac OS X applications must pass UPPs instead of procedure pointers for Open Transport callback routines. You can use these new functions to create UPPs:

OTNotifyUPP replaces OTNotifyProcPtr

OTProcessUPP replaces OTNotifyProcPtr

OTListSearchUPP replaces OTListSearchProcPtr

You can use these functions to allocate and free UPPs:

- NewOTNotifyUPP (page 2305)
- DisposeOTNotifyUPP (page 2302)
- NewOTProcessUPP (page 2306)
- DisposeOTProcessUPP (page 2303)
- NewOTListSearchUPP (page 2305)
- DisposeOTListSearchUPP (page 2302)

These functions have been modified to take an OTNotifyUPP UPP instead of a procedure pointer:

- OTAsyncOpenAppleTalkServicesInContext (page 2310)
- OTAsyncOpenInternetServicesInContext (page 2311)
- OTInstallNotifier (page 2361)
- OTAsyncOpenEndpointInContext (page 2310)
- OTAsyncOpenMapperInContext (page 2312)

These functions have been modified to take an OTProcessUPP UPP instead of a procedure pointer:

- OTCreateTimerTaskInContext (page 2330)
- OTCreateDeferredTaskInContext (page 2329)

These functions have been modified to take an OTListSearchUPP UPP instead of a procedure pointer:

- OTFindLink (page 2339)
- OTFindAndRemoveLink (page 2338)

Functions by Task

Initializing and Closing Open Transport

`CloseOpenTransportInContext` (page 2301) Deprecated in Mac OS X v10.4
Unregisters your application or code resource connection to Open Transport.

`InitOpenTransportInContext` (page 2303) Deprecated in Mac OS X v10.4
Initializes the parts of Open Transport for use by the application or code resource.

Creating, Cloning, and Disposing of a Configuration Structure

`OTCloneConfiguration` (page 2322) Deprecated in Mac OS X v10.4
Copies an `OTConfiguration` structure.

`OTCreateConfiguration` (page 2328) Deprecated in Mac OS X v10.4
Creates a structure defining a provider's configuration.

`OTDestroyConfiguration` (page 2333) Deprecated in Mac OS X v10.4
Deletes an `OTConfiguration` structure.

Opening and Closing Providers

`OTCloseProvider` (page 2323) Deprecated in Mac OS X v10.4
Closes a provider of any type—endpoint, mapper, or service provider.

Controlling a Provider's Modes of Operation

`OTAckSends` (page 2307) Deprecated in Mac OS X v10.4
Specifies that a provider make an internal copy of data being sent and that it notify you when it has finished sending data.

`OTCancelSynchronousCalls` (page 2321) Deprecated in Mac OS X v10.4
Cancels any currently executing synchronous function for a specified provider.

`OTDontAckSends` (page 2335) Deprecated in Mac OS X v10.4
Specifies that a provider copy data before sending it.

`OTIsAckingSends` (page 2363) Deprecated in Mac OS X v10.4
Determines whether a provider is acknowledging sends.

`OTIsSynchronous` (page 2364) Deprecated in Mac OS X v10.4
Returns a provider's current mode of execution.

`OTSetAsynchronous` (page 2390) Deprecated in Mac OS X v10.4
Sets a provider's mode of execution to asynchronous.

`OTSetBlocking` (page 2391) Deprecated in Mac OS X v10.4
Allows a provider to wait or block until it is able to send or receive data.

`OTSetNonBlocking` (page 2396) Deprecated in Mac OS X v10.4
Disallows a provider from waiting if it cannot currently complete a function that sends or receives data.

`OTSetSynchronous` (page 2397) Deprecated in Mac OS X v10.4
Sets a provider's mode of execution to synchronous.

Using Notifier Functions with Providers

`OTEnterNotifier` (page 2336) Deprecated in Mac OS X v10.4
Limits the notifications that can be sent to your notifier.

`OTInstallNotifier` (page 2361) Deprecated in Mac OS X v10.4
Installs a notifier function.

`OTLeaveNotifier` (page 2364) Deprecated in Mac OS X v10.4
Allows Open Transport to resume sending primary and completion events.

`OTRemoveNotifier` (page 2386) Deprecated in Mac OS X v10.4
Removes a provider's notifier function.

`OTUseSyncIdleEvents` (page 2406) Deprecated in Mac OS X v10.4
Allows synchronous idle events to be sent to your notifier.

Sending Module-Specific Commands to Providers

`OTIoctl` (page 2362) Deprecated in Mac OS X v10.4
Sends a module-specific command to an Open Transport protocol module.

Creating Endpoints

`OTAsyncOpenEndpointInContext` (page 2310) Deprecated in Mac OS X v10.4
Opens an endpoint and installs a notifier callback function for the endpoint.

`OTOpenEndpointInContext` (page 2373) Deprecated in Mac OS X v10.4
Opens an endpoint that operates synchronously.

Binding and Unbinding Endpoints

`OTBind` (page 2319) Deprecated in Mac OS X v10.4
Assigns an address to an endpoint.

`OTUnbind` (page 2405) Deprecated in Mac OS X v10.4
Dissociates an endpoint from its address or cancels an asynchronous call to the `OTBind` function.

Obtaining Information About an Endpoint

`OTGetEndpointInfo` (page 2344) Deprecated in Mac OS X v10.4
Obtains information about an endpoint that has been opened.

`OTGetEndpointState` (page 2345) Deprecated in Mac OS X v10.4
Obtains the current state of an endpoint.

`OTGetProtAddress` (page 2348) Deprecated in Mac OS X v10.4
Obtains the address to which an endpoint is bound and, if the endpoint is currently connected, obtains the address of its peer.

`OTLook` (page 2367) Deprecated in Mac OS X v10.4
Determines the current asynchronous event pending for an endpoint.

`OTResolveAddress` (page 2387) Deprecated in Mac OS X v10.4
Returns the protocol address that corresponds to the name of an endpoint.

Allocating Structures for Endpoints

`OTFree` (page 2341) Deprecated in Mac OS X v10.4
Frees memory allocated using the OTAlloc function.

Determining if Bytes Are Available for Endpoints

`OTCountDataBytes` (page 2327) Deprecated in Mac OS X v10.4
Returns the amount of data currently available to be read.

Functions for Connectionless Transactionless Endpoints

`OTRcvUData` (page 2381) Deprecated in Mac OS X v10.4
Reads data sent by a client using a connectionless transactionless protocol.

`OTRcvUDErr` (page 2382) Deprecated in Mac OS X v10.4
Clears an error condition indicated by a `T_UDERR` event and returns the reason for the error.

`OTSndUData` (page 2400) Deprecated in Mac OS X v10.4
Sends data using a connectionless transactionless endpoint.

Establishing Connection for Endpoints

`OTAccept` (page 2306) Deprecated in Mac OS X v10.4
Accepts an incoming connection request.

`OTConnect` (page 2326) Deprecated in Mac OS X v10.4
Requests a connection to a remote peer.

`OTListen` (page 2366) Deprecated in Mac OS X v10.4
Listens for an incoming connection request.

`OTRcvConnect` (page 2379) Deprecated in Mac OS X v10.4
Reads the status of an outstanding or completed asynchronous call to the`OTConnect` function.

Functions for Connection-Oriented Transactionless Endpoints

`OTRcv` (page 2377) Deprecated in Mac OS X v10.4
Reads data sent using a connection-oriented transactionless protocol.

`OTSnd` (page 2397) Deprecated in Mac OS X v10.4
Sends data to a remote peer.

Tearing Down an Endpoint Connection

`OTRcvDisconnect` (page 2379) Deprecated in Mac OS X v10.4
Identifies the cause of a connection break or of a connection rejection, acknowledges and clears the corresponding disconnection event.

`OTRcvOrderlyDisconnect` (page 2380) Deprecated in Mac OS X v10.4
Acknowledges a request for an orderly disconnect.

`OTSndDisconnect` (page 2399) Deprecated in Mac OS X v10.4
Tears down an open connection (abortive disconnect) or rejects an incoming connection request.

`OTSndOrderlyDisconnect` (page 2399) Deprecated in Mac OS X v10.4
Initiates or completes an orderly disconnection.

Checking Synchronous Calls

`OTCanMakeSyncCall` (page 2322) Deprecated in Mac OS X v10.4
Checks whether you can call a synchronous function.

Working With Timer Tasks

`OTCancelTimerTask` (page 2321) Deprecated in Mac OS X v10.4
Cancels a task that was already scheduled for execution.

`OTCreateTimerTaskInContext` (page 2330) Deprecated in Mac OS X v10.4
Creates a task to be scheduled.

`OTDestroyTimerTask` (page 2334) Deprecated in Mac OS X v10.4
Disposes of a timer task.

`OTScheduleTimerTask` (page 2389) Deprecated in Mac OS X v10.4
Schedules a timer task to be executed at the specified time.

Working With Deferred Tasks

`OTCreateDeferredTaskInContext` (page 2329) Deprecated in Mac OS X v10.4
Creates a reference to a task that can be scheduled to run at deferred task time.

`OTDestroyDeferredTask` (page 2334) Deprecated in Mac OS X v10.4
Destroys a deferred task created with the `OTCreateDeferredTask` function.

`OTScheduleDeferredTask` (page 2388) Deprecated in Mac OS X v10.4
Schedules a task for execution at deferred task time.

Creating Mappers

`OTAsyncOpenMapperInContext` (page 2312) Deprecated in Mac OS X v10.4
Creates an asynchronous mapper and installs a notifier function for the mapper provider.

`OTOpenMapperInContext` (page 2374) Deprecated in Mac OS X v10.4
Creates a synchronous mapper provider and returns a mapper reference.

Registering and Deleting Names with Mappers

`OTDeleteName` (page 2331) Deprecated in Mac OS X v10.4
Removes a previously registered entity name.

`OTDeleteNameByID` (page 2332) Deprecated in Mac OS X v10.4
Removes a previously registered name as specified by its name ID.

`OTRegisterName` (page 2384) Deprecated in Mac OS X v10.4
Registers an entity name on the network.

Looking Up Names for Mappers

`OTLookupName` (page 2368) Deprecated in Mac OS X v10.4
Finds and returns all addresses that correspond to a particular name or name pattern, or confirms that a name is registered.

Determining and Changing Option Values

`OTOptionManagement` (page 2375) Deprecated in Mac OS X v10.4
Determines an endpoint's current or default option values or changes these values.

Finding Options

`OTFindOption` (page 2339) Deprecated in Mac OS X v10.4
Finds a specific option in an options buffer.

`OTNextOption` (page 2371) Deprecated in Mac OS X v10.4
Locates the next `TOption` structure in a buffer.

Getting Information About Ports

`OTFindPort` (page 2340) Deprecated in Mac OS X v10.4
Obtains information about a port that corresponds to a given port name.

`OTFindPortByRef` (page 2341) Deprecated in Mac OS X v10.4
Obtains information about a port that corresponds to its given port reference.

`OTGetBusTypeFromPortRef` (page 2343) Deprecated in Mac OS X v10.4
Extracts the value of the bus type from a port reference.

`OTGetDeviceTypeFromPortRef` (page 2343) Deprecated in Mac OS X v10.4
Extracts the value of the hardware device type from a port reference.

`OTGetIndexedPort` (page 2346) Deprecated in Mac OS X v10.4
Iterates through the ports available on your computer.

`OTGetSlotFromPortRef` (page 2349) Deprecated in Mac OS X v10.4
Extracts slot information from a port reference.

Registering New Ports

`OTCreatePortRef` (page 2329) Deprecated in Mac OS X v10.4
Creates a port reference that describes a port's hardware characteristics.

Registering as a Client

`OTRegisterAsClientInContext` (page 2383) Deprecated in Mac OS X v10.4
Registers your application as a client of Open Transport and gives Open Transport a notifier function it can use to send you events.

`OTUnregisterAsClientInContext` (page 2405) Deprecated in Mac OS X v10.4
Removes your application as a client of Open Transport.

Allocating and Freeing Memory

`OTAllocMemInContext` (page 2309) Deprecated in Mac OS X v10.4
Allocates memory using an explicit client context.

`OTFreeMem` (page 2342) Deprecated in Mac OS X v10.4
Frees memory allocated with the `OTAllocMem` function.

Memory Manipulation Utility Functions

`OTMemcmp` (page 2369) Deprecated in Mac OS X v10.4
Compares the contents of two memory locations.

`OTMemcpy` (page 2370) Deprecated in Mac OS X v10.4
Copies data from one memory location to another; the source and destination locations must not overlap.

`OTMemmove` (page 2370) Deprecated in Mac OS X v10.4
Copies data from one memory location to another; the source and destination locations may overlap.

`OTMemset` (page 2370) Deprecated in Mac OS X v10.4
Sets the specified memory range to a specific value.

`OTMemzero` (page 2371) Deprecated in Mac OS X v10.4
Initializes the specified memory range to 0.

Idling and Delaying Processing

`OTDelay` (page 2331) Deprecated in Mac OS X v10.4
Delays processing for a specified number of seconds. This function is only provided for compatibility with the UNIX `sleep` function.

`OTIdle` (page 2350) Deprecated in Mac OS X v10.4
Idles your computer.

String Manipulation Utility Functions

`OTStrCat` (page 2401) Deprecated in Mac OS X v10.4
Concatenates two C strings.

`OTStrCopy` (page 2402) Deprecated in Mac OS X v10.4
Copies a C string.

`OTStrEqual` (page 2402) Deprecated in Mac OS X v10.4
Determines whether two C strings are the same.

`OTStrLength` (page 2402) Deprecated in Mac OS X v10.4
Returns the length of a C string.

Timestamp Utility Functions

`OTElapsedMicroseconds` (page 2335) Deprecated in Mac OS X v10.4
Calculates the time elapsed in microseconds since a specified time.

`OTElapsedMilliseconds` (page 2335) Deprecated in Mac OS X v10.4
Calculates the time elapsed in milliseconds since a specified time.

`OTGetClockTimeInSecs` (page 2343) Deprecated in Mac OS X v10.4
Returns the number of seconds that have elapsed since system boot time.

`OTGetTimeStamp` (page 2350) Deprecated in Mac OS X v10.4
Obtains the current timestamp.

`OTSubtractTimeStamps` (page 2403) Deprecated in Mac OS X v10.4
Subtracts one timestamp value from another.

`OTTimeStampInMicroseconds` (page 2404) Deprecated in Mac OS X v10.4
Calculates the time elapsed in microseconds since since a specified time.

`OTTimeStampInMilliseconds` (page 2404) Deprecated in Mac OS X v10.4
Calculates the time elapsed in milliseconds since since a specified time.

OTLIFO List Utility Functions

`OTLIFODequeue` (page 2365) Deprecated in Mac OS X v10.4
Removes the first link in a LIFO list and returns a pointer to it.

`OTLIFOEnqueue` (page 2365) Deprecated in Mac OS X v10.4
Places a link at the front of a LIFO list.

`OTLIFOStealList` (page 2366) Deprecated in Mac OS X v10.4
Removes all links in a LIFO list and returns a pointer to the first link in the list.

`OTReverseList` (page 2387) Deprecated in Mac OS X v10.4
Reverses the order in which entries are linked in a list.

OTFIFO List Utility Functions

`OTAddFirst` (page 2308) Deprecated in Mac OS X v10.4
Places a link at the front of a FIFO list.

`OTAddLast` (page 2308) Deprecated in Mac OS X v10.4
Adds a link to the end of a FIFO list.

`OTFindAndRemoveLink` (page 2338) Deprecated in Mac OS X v10.4
Finds a link in a FIFO list and removes it.

`OTFindLink` (page 2339) Deprecated in Mac OS X v10.4
Finds a link in a FIFO list and returns a pointer to it.

`OTGetFirst` (page 2345) Deprecated in Mac OS X v10.4
Returns a pointer to the first element in a FIFO list.

`OTGetIndexedLink` (page 2346) Deprecated in Mac OS X v10.4
Returns a pointer to the link at a specified position in a FIFO list.

`OTGetLast` (page 2347) Deprecated in Mac OS X v10.4
Returns the last element in a FIFO list.

`OTIsInList` (page 2363) Deprecated in Mac OS X v10.4
Determines whether the specified link is in the specified list.

`OTRemoveFirst` (page 2385) Deprecated in Mac OS X v10.4
Removes the first link in a FIFO list.

`OTRemoveLast` (page 2385) Deprecated in Mac OS X v10.4
Removes the last link in a FIFO list.

`OTRemoveLink` (page 2386) Deprecated in Mac OS X v10.4
Removes the last link in a FIFO list.

Adding and Removing List Elements

`OTDequeue` (page 2333) Deprecated in Mac OS X v10.4
Removes an element from a list.

`OTEnqueue` (page 2336) Deprecated in Mac OS X v10.4
Adds an element to a list.

Atomic Operations

`OTAtomicAdd16` (page 2316) Deprecated in Mac OS X v10.4
Atomically adds a 16-bit value to a memory location.

`OTAtomicAdd32` (page 2316) Deprecated in Mac OS X v10.4
Atomically adds a 32-bit value to a memory location.

`OTAtomicAdd8` (page 2317) Deprecated in Mac OS X v10.4
Atomically adds an 8-bit value to a memory location.

`OTAtomicClearBit` (page 2317) Deprecated in Mac OS X v10.4
Clears a bit in a byte.

`OTAtomicSetBit` (page 2318) Deprecated in Mac OS X v10.4
Sets a specified bit in a byte.

`OTAtomicTestBit` (page 2318) Deprecated in Mac OS X v10.4
Tests a bit in a byte and returns its current state.

`OTCompareAndSwap16` (page 2324) Deprecated in Mac OS X v10.4
Atomically compares two 16-bit values and changes one of these values if they are the same.

`OTCompareAndSwap32` (page 2324) Deprecated in Mac OS X v10.4
Atomically compares two 32-bit values and changes one of these values if they are the same.

`OTCompareAndSwap8` (page 2325) Deprecated in Mac OS X v10.4
Atomically compares two 8-bit values and changes one of these values if they are the same.

`OTCompareAndSwapPtr` (page 2325) Deprecated in Mac OS X v10.4
Atomically compares the value of a pointer at a memory location and atomically swaps it with a second pointer value if the compare is successful.

Handling No-Copy Receives

`OTBufferDataSize` (page 2320) Deprecated in Mac OS X v10.4
Obtains the size of the no-copy receive buffer.

`OTReadBuffer` (page 2383) Deprecated in Mac OS X v10.4
Copies data out of a no-copy receive buffer.

`OTReleaseBuffer` (page 2384) Deprecated in Mac OS X v10.4
Returns the no-copy receive buffer to the system.

Resolving Internet Addresses

`OTInetAddressToName` (page 2350) Deprecated in Mac OS X v10.4
Determines the canonical domain name of the host associated with an internet address.

`OTInetStringToAddress` (page 2355) Deprecated in Mac OS X v10.4
Resolves a domain name to its equivalent internet addresses.

Opening a TCP/IP Service Provider

`OTAsyncOpenInternetServicesInContext` (page 2311) Deprecated in Mac OS X v10.4
Opens the TCP/IP service provider and returns an Internet services reference.

`OTOpenInternetServicesInContext` (page 2374) Deprecated in Mac OS X v10.4
Opens the TCP/IP service provider and returns an internet services reference.

Getting Information About an Internet Host

`OTInetMailExchange` (page 2353) Deprecated in Mac OS X v10.4
Returns mail-exchange-host names and preference information for a domain name you specify.

`OTInetSysInfo` (page 2356) Deprecated in Mac OS X v10.4
Returns details about a host's processor and operating system.

Retrieving DNS Query Information

`OTInetQuery` (page 2354) Deprecated in Mac OS X v10.4
Executes a generic DNS query.

Internet Address Utilities

`OTInetGetInterfaceInfo` (page 2351) Deprecated in Mac OS X v10.4
Returns internet address information about the local host.

`OTInetHostToString` (page 2352) Deprecated in Mac OS X v10.4
Converts an an address in InetHost format into a character string in dotted-decimal notation.

`OTInetStringToHost` (page 2356) Deprecated in Mac OS X v10.4
Converts an IP address string from dotted-decimal notation or hexadecimal notation to an `InetHost` data type.

`OTInitDNSAddress` (page 2358) Deprecated in Mac OS X v10.4
Fills in a DNSAddress structure with the data you provide.

`OTInitInetAddress` (page 2359) Deprecated in Mac OS X v10.4
Fills in an InetAddress structure with the data you provide.

Single Link Multi-Homing

`OTInetGetSecondaryAddresses` (page 2352) Deprecated in Mac OS X v10.4
Returns the active secondary IP addresses.

AppleTalk Utility Functions

`OTCompareDDPAddresses` (page 2325) Deprecated in Mac OS X v10.4
Compares two DDP address structures.

`OTExtractNBPName` (page 2337) Deprecated in Mac OS X v10.4
Extracts the name part of an NBP name from an NBP entity structure.

`OTExtractNBPType` (page 2337) Deprecated in Mac OS X v10.4
Extracts the type part of an NBP name from an NBP entity structure.

`OTExtractNBPZone` (page 2338) Deprecated in Mac OS X v10.4
Extracts the zone part of an NBP name from an NBP entity structure.

`OTGetNBPEntityLengthAsAddress` (page 2348) Deprecated in Mac OS X v10.4
Obtains the size of an NBP entity structure.

`OTInitDDPAddress` (page 2357) Deprecated in Mac OS X v10.4
Initializes a DDP address structure.

`OTInitDDPNBPAddress` (page 2358) Deprecated in Mac OS X v10.4
Initializes a combined DDP-NBP address structure.

`OTInitNBPAddress` (page 2360) Deprecated in Mac OS X v10.4
Initializes an NBP address structure.

`OTInitNBPEntity` (page 2360) Deprecated in Mac OS X v10.4
Initializes an NBP entity structure.

`OTSetAddressFromNBPEntity` (page 2389) Deprecated in Mac OS X v10.4
Stores an NBP entity structure as an NBP address string.

`OTSetAddressFromNBPString` (page 2390) Deprecated in Mac OS X v10.4
Copies an NBP name string into an NBP address buffer.

`OTSetNBPEntityFromAddress` (page 2394) Deprecated in Mac OS X v10.4
Parses and stores an NBP address into an NBP entity.

`OTSetNBPName` (page 2394) Deprecated in Mac OS X v10.4
Stores the name part of an NBP name into an NBP entity structure.

`OTSetNBPType` (page 2395) Deprecated in Mac OS X v10.4
Stores the type part of an NBP name in an NBP entity structure.

`OTSetNBPZone` (page 2396) Deprecated in Mac OS X v10.4
Stores the zone part of an NBP name in an NBP entity structure.

Opening an AppleTalk Service Provider

`OTAsyncOpenAppleTalkServicesInContext` (page 2310) Deprecated in Mac OS X v10.4
Opens an asynchronous AppleTalk service provider in context.

`OTOpenAppleTalkServicesInContext` (page 2372) Deprecated in Mac OS X v10.4
Opens a synchronous AppleTalk service provider.

Obtaining Information About Zones

`OTATalkGetLocalZones` (page 2314) Deprecated in Mac OS X v10.4
Obtains a list of the zones available on your network.

`OTATalkGetMyZone` (page 2314) Deprecated in Mac OS X v10.4
Obtains the AppleTalk zone name of the node on which your application is running.

`OTATalkGetZoneList` (page 2315) Deprecated in Mac OS X v10.4
Obtains a list of all the zones available on the AppleTalk internet.

Obtaining Information About Your AppleTalk Environment

`OTATalkGetInfo` (page 2313) Deprecated in Mac OS X v10.4
Obtains information about the AppleTalk environment for a given node.

Miscellaneous Functions

`DisposeOTListSearchUPP` (page 2302) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a list search callback.

`DisposeOTNotifyUPP` (page 2302) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a notification callback.

`DisposeOTProcessUPP` (page 2303) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a process callback.

`InvokeOTListSearchUPP` (page 2304) Deprecated in Mac OS X v10.4
Calls a list search callback.

`InvokeOTNotifyUPP` (page 2304) Deprecated in Mac OS X v10.4
Calls a notification callback.

`InvokeOTProcessUPP` (page 2305) Deprecated in Mac OS X v10.4
Calls a process callback.

`NewOTListSearchUPP` (page 2305) Deprecated in Mac OS X v10.4
Creates a new universal procedure pointer (UPP) to a list search callback.

`NewOTNotifyUPP` (page 2305) Deprecated in Mac OS X v10.4
Creates a new universal procedure pointer (UPP) to a notification callback.

`NewOTProcessUPP` (page 2306) Deprecated in Mac OS X v10.4
Creates a new universal procedure pointer (UPP) to a process callback.

`OTAllocInContext` (page 2308) Deprecated in Mac OS X v10.4
Allocates a data structure of a specified type.

`OTClearBit` (page 2322) Deprecated in Mac OS X v10.4
Clears a bit atomically.

`OTIsBlocking` (page 2363) Deprecated in Mac OS X v10.4
Returns a boolean indicating whether a provider is blocking.

`OTSetBit` (page 2391) Deprecated in Mac OS X v10.4
Sets a bit atomically.

`OTSetBusTypeInPortRef` (page 2392) Deprecated in Mac OS X v10.4
Sets bus type for a port reference.

`OTSetDeviceTypeInPortRef` (page 2393) Deprecated in Mac OS X v10.4
Sets device type for a port reference.

`OTSetFirstClearBit` (page 2393) Deprecated in Mac OS X v10.4
Atomcially sets the first clear bit in a specified bit map.

`OTTestBit` (page 2403) Deprecated in Mac OS X v10.4
Atomically tests a bit in a specified bit map.

Functions

CloseOpenTransportInContext

Unregisters your application or code resource connection to Open Transport. (Deprecated in Mac OS X v10.4.)

```
void CloseOpenTransportInContext (
   OTClientContextPtr clientContext
);
```

Parameters

clientContext

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Carbon Porting Notes

The `CloseOpenTransportinContext` function acts like the pre-Carbon `CloseOpenTransport` function except that it takes an additional parameter, an `OTClientContextPtr`, which can be `NULL` for applications. Other types of clients must provide a valid client context pointer.

Declared In

`OpenTransport.h`

DisposeOTListSearchUPP

Disposes of a universal procedure pointer (UPP) to a list search callback. (Deprecated in Mac OS X v10.4.)

```
void DisposeOTListSearchUPP (
   OTListSearchUPP userUPP
);
```

Parameters

userUPP

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`OpenTransport.h`

DisposeOTNotifyUPP

Disposes of a universal procedure pointer (UPP) to a notification callback. (Deprecated in Mac OS X v10.4.)

```
void DisposeOTNotifyUPP (
   OTNotifyUPP userUPP
);
```

Parameters

userUPP

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
`OpenTransport.h`

DisposeOTProcessUPP

Disposes of a universal procedure pointer (UPP) to a process callback. (Deprecated in Mac OS X v10.4.)

```
void DisposeOTProcessUPP (
   OTProcessUPP userUPP
);
```

Parameters

userUPP

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`OpenTransport.h`

InitOpenTransportInContext

Initializes the parts of Open Transport for use by the application or code resource. (Deprecated in Mac OS X v10.4.)

```
OSStatus InitOpenTransportInContext (
   OTInitializationFlags flags,
   OTClientContextPtr *outClientContext
);
```

Parameters

flags
Tells Open Transport whether your code is an application or a plug-in.

outClientContext
Returns the client context pointer.

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
In Carbon, the `InitOpenTransportInContext` function acts like the pre-Carbon InitOpenTransport function, except that it takes parameters that specify initialization context explicitly.

Use the flags parameter to tell Open Transport whether your code is an application or some other target (for example, a plug-in that runs in an application context but is not the application itself). The second parameter returns the client context pointer, which you must pass to other asset-creation routines. For more information, see Understanding Open Transport Asset Tracking at http://developer.apple.com/technotes/tn/tn1173.html.

Availability
Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransport.h`

InvokeOTListSearchUPP

Calls a list search callback. (Deprecated in Mac OS X v10.4.)

```
Boolean InvokeOTListSearchUPP (
   const void *ref,
   OTLink *linkToCheck,
   OTListSearchUPP userUPP
);
```

Parameters
ref
linkToCheck
userUPP

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`OpenTransport.h`

InvokeOTNotifyUPP

Calls a notification callback. (Deprecated in Mac OS X v10.4.)

```
void InvokeOTNotifyUPP (
   void *contextPtr,
   OTEventCode code,
   OTResult result,
   void *cookie,
   OTNotifyUPP userUPP
);
```

Parameters
contextPtr
code
result
cookie
userUPP

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`OpenTransport.h`

InvokeOTProcessUPP

Calls a process callback. (Deprecated in Mac OS X v10.4.)

```
void InvokeOTProcessUPP (
   void *arg,
   OTProcessUPP userUPP
);
```

Parameters

arg

userUPP

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`OpenTransport.h`

NewOTListSearchUPP

Creates a new universal procedure pointer (UPP) to a list search callback. (Deprecated in Mac OS X v10.4.)

```
OTListSearchUPP NewOTListSearchUPP (
   OTListSearchProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `OTListSearchUPP` data type.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`OpenTransport.h`

NewOTNotifyUPP

Creates a new universal procedure pointer (UPP) to a notification callback. (Deprecated in Mac OS X v10.4.)

```
OTNotifyUPP NewOTNotifyUPP (
   OTNotifyProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `OTNotifyUPP` data type.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`OpenTransport.h`

NewOTProcessUPP

Creates a new universal procedure pointer (UPP) to a process callback. (Deprecated in Mac OS X v10.4.)

```
OTProcessUPP NewOTProcessUPP (
   OTProcessProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `OTProcessUPP` data type.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`OpenTransport.h`

OTAccept

Accepts an incoming connection request. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTAccept (
   EndpointRef listener,
   EndpointRef worker,
   TCall *call
);
```

Parameters

listener

worker

call

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTAckSends

Specifies that a provider make an internal copy of data being sent and that it notify you when it has finished sending data. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTAckSends (
   ProviderRef ref
);
```

Parameters

ref

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
By default, providers make an internal copy of data before sending it and they do not acknowledge sends. If you use the `OTAckSends` function to specify that the provider acknowledge sends and you call a function that sends data, the provider does not copy the data before sending it. Instead, it reads data directly from your buffer while sending. For this reason, you must not change the contents of your buffer until the provider is no longer using it. The provider lets you know that it has finished using the buffer by calling your notifier function and passing `T_MEMORYRELEASED` event code for the `code` parameter, a pointer to the buffer that was sent in the `cookie` parameter, and the size of the buffer in the `result` parameter.

If you have not installed a notifier function for the provider, this function returns the `kOTAccessErr` result.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTAddFirst

Places a link at the front of a FIFO list. (Deprecated in Mac OS X v10.4.)

```
void OTAddFirst (
   OTList *list,
   OTLink *link
);
```

Parameters

list

link

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTAddLast

Adds a link to the end of a FIFO list. (Deprecated in Mac OS X v10.4.)

```
void OTAddLast (
   OTList *list,
   OTLink *link
);
```

Parameters

list

link

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTAllocInContext

Allocates a data structure of a specified type. (Deprecated in Mac OS X v10.4.)

```
void * OTAllocInContext (
   EndpointRef ref,
   OTStructType structType,
   UInt32 fields,
   OSStatus *err,
   OTClientContextPtr clientContext
);
```

Parameters

ref

structType

fields

err

clientContext

Discussion

In general, Apple recommends that you avoid the OTAllocInContext call because using it extensively causes your program to allocate and deallocate many memory blocks, with each extra memory allocation costing time.

Under Carbon, OTAllocInContext takes a client context pointer. Applications may pass NULL after calling InitOpenTransport(kInitOTForApplicationMask, ...). Non-applications must always pass a valid client context.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

OpenTransport.h

OTAllocMemInContext

Allocates memory using an explicit client context. (Deprecated in Mac OS X v10.4.)

```
void * OTAllocMemInContext (
   OTByteCount size,
   OTClientContextPtr clientContext
);
```

Parameters

size

clientContext

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

OpenTransport.h

OTAsyncOpenAppleTalkServicesInContext

Opens an asynchronous AppleTalk service provider in context. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTAsyncOpenAppleTalkServicesInContext (
   OTConfigurationRef cfig,
   OTOpenFlags flags,
   OTNotifyUPP proc,
   void *contextPtr,
   OTClientContextPtr clientContext
);
```

Parameters

cfig

flags

proc

contextPtr

clientContext

Return Value

A result code. See "Open Transport Result Codes" (page 2722).

Discussion

Applications may pass a NULL context pointer but nonapplications must always pass a valid client context pointer.

You receive a client context pointer when you call the function `InitOpenTransportInContext` (page 2303).

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`OpenTransportProviders.h`

OTAsyncOpenEndpointInContext

Opens an endpoint and installs a notifier callback function for the endpoint. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTAsyncOpenEndpointInContext (
   OTConfigurationRef config,
   OTOpenFlags oflag,
   TEndpointInfo *info,
   OTNotifyUPP upp,
   void *contextPtr,
   OTClientContextPtr clientContext
);
```

Parameters

config

oflag

info

upp

contextPtr

clientContext

Return Value

A result code. See "Open Transport Result Codes" (page 2722).

Discussion

Applications may pass a NULL context pointer but nonapplications must always pass a valid client context pointer.

You receive a client context pointer when you call the function `InitOpenTransportInContext` (page 2303).

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`OpenTransport.h`

OTAsyncOpenInternetServicesInContext

Opens the TCP/IP service provider and returns an Internet services reference. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTAsyncOpenInternetServicesInContext (
   OTConfigurationRef cfig,
   OTOpenFlags oflag,
   OTNotifyUPP upp,
   void *contextPtr,
   OTClientContextPtr clientContext
);
```

Parameters

cfig

oflag

upp

contextPtr

clientContext

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
Applications may pass a NULL context pointer but nonapplications must always pass a valid client context pointer.

You receive a client context pointer when you call the function `InitOpenTransportInContext` (page 2303).

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTAsyncOpenMapperInContext

Creates an asynchronous mapper and installs a notifier function for the mapper provider. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTAsyncOpenMapperInContext (
   OTConfigurationRef config,
   OTOpenFlags oflag,
   OTNotifyUPP upp,
   void *contextPtr,
   OTClientContextPtr clientContext
);
```

Parameters

config

oflag

upp

contextPtr

clientContext

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
Applications may pass a NULL context pointer but nonapplications must always pass a valid client context pointer.

You receive a client context pointer when you call the function `InitOpenTransportInContext` (page 2303).

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransport.h`

OTATalkGetInfo

Obtains information about the AppleTalk environment for a given node. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTATalkGetInfo (
   ATSvcRef ref,
   TNetbuf *info
);
```

Parameters
ref
info

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
The `OTATalkGetInfo` function returns the information contained in the `AppleTalkInfo` data structure that describes your current AppleTalk environment. This includes your network number and node ID, the network number and node ID of a local router, and the current network range for the extended network to which the machine is connected.

If you execute this function asynchronously, Open Transport calls your notifier with a `T_GETATALKINFOCOMPLETE` completion event to signal the function's completion and uses your notifier's `cookie` parameter for the AppleTalk information. The `cookie` parameter actually holds a pointer to a `TNetbuf` structure, which points in turn to a buffer containing the `AppleTalkInfo` structure. The maximum size of this buffer is 22 bytes.

If the machine is multihomed—that is, if multiple network numbers and node numbers are associated with the same machine—the `OTATalkGetInfo` function returns information about the node whose network number and node ID are selected in the AppleTalk control panel.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTATalkGetLocalZones

Obtains a list of the zones available on your network. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTATalkGetLocalZones (
   ATSvcRef ref,
   TNetbuf *zones
);
```

Parameters

ref

zones

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
The `OTATalkGetLocalZones` function returns a list of the zone names in your application's network if it is an extended network. These are all the zones to which your node can belong. If your application is in a nonextended network, this function returns only one zone name, the same one returned by the `OTATalkGetMyZone` function.

If you execute this function asynchronously, Open Transport calls your notifier function with a `T_GETLOCALZONESCOMPLETE` completion event to signal the function's completion and uses your notifier's `cookie` parameter for the list of zones. The `cookie` parameter actually holds a pointer to a `TNetbuf` structure, which points to a buffer containing a list of zone names, each of which is stored as a Pascal-style string. Using a Pascal-style string for the zone name is redundant since you can determine the length of the string from the `maxlen` field of the `TNetbuf` structure, but the other zone-related calls use Pascal-style strings, so this call also uses them for consistency.

Each string can be up to 32 characters in length, and if you add a length byte, each can have a maximum size of 33 bytes. As there can be a maximum of 254 zones on an extended network, the maximum size of the buffer is 8382 bytes.

Because zone names are often less than 32 characters long and AppleTalk service providers don't pad short names, 6 KB bytes is likely to be a safe value for the buffer's size, defined by the `TNetbuf->maxlen` field.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTATalkGetMyZone

Obtains the AppleTalk zone name of the node on which your application is running. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTATalkGetMyZone (
   ATSvcRef ref,
   TNetbuf *zone
);
```

Parameters

ref

zone

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
The `OTATalkGetMyZone` function gets the name of your application's AppleTalk zone. If you call this function asynchronously, Open Transport calls your application's notifier with a `T_GETMYZONECOMPLETE` completion event to signal the function's completion and uses your notifier's `cookie` parameter for the zone name. More precisely, the `cookie` parameter points to a `TNetbuf` structure that in turn points to a buffer containing the zone name, which is stored as a Pascal-style string. The string can be up to 32 characters in length, so with the addition of a length byte, the buffer can have a maximum size of 33 bytes. Using a Pascal-style string for the zone name is redundant since you can determine the length of the string from the `maxlen` field of the `TNetbuf` structure, but the other zone-related calls use Pascal-style strings, so this call also uses them for consistency.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTATalkGetZoneList

Obtains a list of all the zones available on the AppleTalk internet. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTATalkGetZoneList (
   ATSvcRef ref,
   TNetbuf *zones
);
```

Parameters

ref

zones

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
The OTATalkGetZoneList function returns a list of all the zones on the AppleTalk internet to which your network belongs.

If you execute this function asynchronously, Open Transport calls your notifier function with a `T_GETZONELISTCOMPLETE` completion event to signal the function's completion and uses your notifier's `cookie` parameter for the list of zones. The `cookie` parameter actually holds a pointer to a `TNetbuf` structure,

which points to a buffer containing a list of zone names, each of which is a Pascal-style string. Using a Pascal-style string for the zone name is redundant since you can determine the length of the string from the `maxlen` field of the `TNetbuf` structure, but the other zone-related calls use Pascal-style strings, so this call also uses them for consistency.

Each string can be up to 32 characters in length, and if you add a length byte, each can have a maximum size of 33 bytes. As AppleTalk internets can have a number of extended networks, you need to allocate a buffer (using the `TNetbuf->maxlen` field) that holds as much as 64 KB of memory. To keep the buffer size as small and efficient as possible, you can set up a large buffer, test for the `kOTBufferOverflowErr` error, and then increase the size of the buffer and reissue the call if this error is returned.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTAtomicAdd16

Atomically adds a 16-bit value to a memory location. (Deprecated in Mac OS X v10.4.)

```
SInt16 OTAtomicAdd16 (
   SInt32 toAdd,
   SInt16 *dest
);
```

Parameters
toAdd
dest

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTAtomicAdd32

Atomically adds a 32-bit value to a memory location. (Deprecated in Mac OS X v10.4.)

```
SInt32 OTAtomicAdd32 (
   SInt32 toAdd,
   SInt32 *dest
);
```

Parameters

toAdd

dest

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTAtomicAdd8

Atomically adds an 8-bit value to a memory location. (Deprecated in Mac OS X v10.4.)

```
SInt8 OTAtomicAdd8 (
   SInt32 toAdd,
   SInt8 *dest
);
```

Parameters

toAdd

dest

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTAtomicClearBit

Clears a bit in a byte. (Deprecated in Mac OS X v10.4.)

```
Boolean OTAtomicClearBit (
   UInt8 *bytePtr,
   OTByteCount bitNumber
);
```

Parameters

bytePtr

bitNumber

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

OpenTransport.h

OTAtomicSetBit

Sets a specified bit in a byte. (Deprecated in Mac OS X v10.4.)

```
Boolean OTAtomicSetBit (
   UInt8 *bytePtr,
   OTByteCount bitNumber
);
```

Parameters

bytePtr

bitNumber

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

OpenTransport.h

OTAtomicTestBit

Tests a bit in a byte and returns its current state. (Deprecated in Mac OS X v10.4.)

```
Boolean OTAtomicTestBit (
   UInt8 *bytePtr,
   OTByteCount bitNumber
);
```

Parameters

bytePtr

bitNumber

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTBind

Assigns an address to an endpoint. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTBind (
   EndpointRef ref,
   TBind *reqAddr,
   TBind *retAddr
);
```

Parameters

ref

reqAddr

If you specify `NIL` for the `reqAddr` parameter, Open Transport chooses a protocol address for you and requests 0 as the endpoint's maximum number of concurrent outstanding connect indications.

If you want Open Transport to assign an address for you, setthe `addr.len` field of the `TBind` structure to 0.

retAddr

You can set this parameter to `nil` if you do not care to know what address the endpoint is bound to or what the negotiated value of `qlen` is.

Return Value

A result code. See "Open Transport Result Codes" (page 2722).

Discussion

You call the `OTBind` function to request an address that an endpoint be bound to. You can either use the reqAddr parameter to request that the endpoint be bound to a specific address or allow the endpoint provider to assign an address dynamically by passing nil for this parameter. Consult the documentation for the top-level protocol you are using to determine whether it is preferable to have the address assigned dynamically. The function returns the address to which the endpoint is actually bound in the retAddr parameter. This might be different from the address you requested, if you requested a specific address.

If you are binding a connection-oriented endpoint, you must use the reqAddr->qlen field to specify the number of connection requests that may be outstanding for this endpoint. The retAddr->qlen field specifies, on return, the actual number of connection requests allowed for the endpoint. This number might be smaller than the number you requested. Note that when the endpoint is actually connected, the number might be further decreased by negotiations taking place at that time.

If you call the OTBind function asynchronously and you have not installed a notifier function, the only way to determine when the function completes is to poll the endpoint using the OTGetEndpointState function. This function returns a kOTStateChangeErr until the bind completes. When the endpoint is bound, the state is either T_UNBND if the bind failed, or T_IDLE if it succeeded.

You can cancel an asynchronous bind that is still in progress by calling the OTUnbind function.

You must not bind more than one connectionless endpoint to a single address. Some connection-oriented protocols let you bind two or more endpoints to the same address. In such instances, you must use only one of the endpoints to listen for connection requests for that address. When binding the endpoint listening for a connection, you must set the reqAddr->qlen field of the OTBind function to a value greater than or equal to 1. When binding the other endpoints, you must set the reqAddr->qlen field to 0.

If you accept a connection for an endpoint that is also listening for connection requests, the address of that endpoint is deemed "busy" for the duration of the connection, and you must not bind another endpoint for listening to that same address. This requirement prevents more than one endpoint bound to the same address from accepting connection requests. If you have to bind another listening endpoint to the same address, you must first use the OTUnbind function to unbind the first endpoint or use the OTCloseProvider function to close it.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTBufferDataSize

Obtains the size of the no-copy receive buffer. (Deprecated in Mac OS X v10.4.)

```
OTByteCount OTBufferDataSize (
   OTBuffer *buffer
);
```

Parameters
buffer

Return Value
See the description of the `OTByteCount` data type.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransportProtocol.h`

OTCancelSynchronousCalls

Cancels any currently executing synchronous function for a specified provider. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTCancelSynchronousCalls (
   ProviderRef ref,
   OSStatus err
);
```

Parameters

ref

err

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
The `OTCancelSynchronousCalls` function cancels any currently executing synchronous function for the provider that you specify. The provider need not be in synchronous mode when you call this function.

Typically, you would call the `OTCancelSynchronousCalls` function at interrupt time by installing a Time Manager task that executes after a given amount of time has passed. You could do this to prevent a synchronous function from hanging the system.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTCancelTimerTask

Cancels a task that was already scheduled for execution. (Deprecated in Mac OS X v10.4.)

```
Boolean OTCancelTimerTask (
   OTTimerTask timerTask
);
```

Parameters

timerTask

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProtocol.h`

OTCanMakeSyncCall

Checks whether you can call a synchronous function. (Deprecated in Mac OS X v10.4.)

```
Boolean OTCanMakeSyncCall (
   void
);
```

Parameters

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTClearBit

Clears a bit atomically. (Deprecated in Mac OS X v10.4.)

```
Boolean OTClearBit (
   UInt8 *bitMap,
   OTByteCount bitNo
);
```

Parameters
bitMap
bitNo

Discussion
OTClearBit is available to client and kernel code, but only to native architecture clients.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransportProtocol.h`

OTCloneConfiguration

Copies an `OTConfiguration` structure. (Deprecated in Mac OS X v10.4.)

```
OTConfigurationRef OTCloneConfiguration (
   OTConfigurationRef cfig
);
```

Parameters

cfig

Return Value
See the description of the `OTConfigurationRef` data type.

Discussion
The `OTCloneConfiguration` function copies the `OTConfiguration` structure that you specify in the `cfig` parameter and returns a pointer to the copy. Because the internal format of an `OTConfiguration` structure is private, you must use the `OTCloneConfiguration` function to obtain two identical structures. For example, you can use this function when another application passes you a configuration structure that you want to reuse but for which you do not have the original configuration string. By cloning the structure, you have access to an additional copy of the configuration even without knowing its configuration string.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTCloseProvider

Closes a provider of any type—endpoint, mapper, or service provider. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTCloseProvider (
   ProviderRef ref
);
```

Parameters

ref

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
The `OTCloseProvider` function closes the provider that you specify in the `ref` parameter. Closing the provider deletes all memory reserved for it in the system heap, deletes its resources, and cancels any provider functions that are currently executing.

Open Transport does not guarantee that all outstanding functions have completed before it closes the provider. It is ultimately your responsibility to make sure that all provider functions that you care about have finished executing, before you close and delete a provider.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
OpenTransport.h

OTCompareAndSwap16

Atomically compares two 16-bit values and changes one of these values if they are the same. (Deprecated in Mac OS X v10.4.)

```
Boolean OTCompareAndSwap16 (
   UInt32 oldValue,
   UInt32 newValue,
   UInt16 *dest
);
```

Parameters

oldValue

newValue

dest

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
OpenTransport.h

OTCompareAndSwap32

Atomically compares two 32-bit values and changes one of these values if they are the same. (Deprecated in Mac OS X v10.4.)

```
Boolean OTCompareAndSwap32 (
   UInt32 oldValue,
   UInt32 newValue,
   UInt32 *dest
);
```

Parameters

oldValue

newValue

dest

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
OpenTransport.h

OTCompareAndSwap8

Atomically compares two 8-bit values and changes one of these values if they are the same. (Deprecated in Mac OS X v10.4.)

```
Boolean OTCompareAndSwap8 (
   UInt32 oldValue,
   UInt32 newValue,
   UInt8 *dest
);
```

Parameters

oldValue

newValue

dest

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

OpenTransport.h

OTCompareAndSwapPtr

Atomically compares the value of a pointer at a memory location and atomically swaps it with a second pointer value if the compare is successful. (Deprecated in Mac OS X v10.4.)

```
Boolean OTCompareAndSwapPtr (
   void *oldValue,
   void *newValue,
   void **dest
);
```

Parameters

oldValue

newValue

dest

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

OpenTransport.h

OTCompareDDPAddresses

Compares two DDP address structures. (Deprecated in Mac OS X v10.4.)

```
Boolean OTCompareDDPAddresses (
   const DDPAddress *addr1,
   const DDPAddress *addr2
);
```

Parameters

addr1

addr2

Discussion

The `OTCompareDDPAddresses` function compares two DDP addresses for equality and returns `true` if the two addresses match. It cannot compare NBP or combined DDP-NBP addresses; using these address types always returns `false`. This function uses the zero-matches-anything AppleTalk rule when doing the matching, which means that a value of 0 in any field results in an acceptable match.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`OpenTransportProviders.h`

OTConnect

Requests a connection to a remote peer. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTConnect (
   EndpointRef ref,
   TCall *sndCall,
   TCall *rcvCall
);
```

Parameters

ref

sndCall

rcvCall

This parameter is only meaningful for synchronous calls to the OTConnect function. See `TCall` data type.

Return Value

A result code. See "Open Transport Result Codes" (page 2722).

Discussion

If the endpoint is in synchronous mode, the OTConnect function returns after the connection is established and fills in the fields of the TCall structure (referenced by the rcvCall parameter) with the actual values associated with this connection. These might be different from the values you specified using the sndCall parameter.

If the OTConnect function returns with the kOTLookErr result, this might be either because of a pending T_LISTEN or T_DISCONNECT event. That is, either a connection request from another endpoint has interrupted execution of the function, or the remote endpoint has rejected the connection. If you don't have a notifier installed, you can call the OTLook function to identify the event that caused the kOTLookErr result. If the

event is T_LISTEN, you must accept or reject the incoming request and then continue processing the OTConnect function by calling OTRcvConnect. If the event is T_DISCONNECT, you must call the OTRcvDisconnect function to clear the error condition—that is, to deallocate memory and place the endpoint in the correct state.

If the endpoint is in asynchronous mode, the OTConnect function returns before the connection is established with a kOTNoDataErr result to indicate that the connection is in progress. When the connection is established, the endpoint provider calls your notifier, passing T_CONNECT for the code parameter. In response, you must call the OTRcvConnect function to read the connection parameters that would have been returned using the rcvCall parameter if the endpoint were in synchronous mode.

It is possible that the remote address returned in the addr field of the rcvCall parameter is not the same as the address you requested using the sndCall->addr field. This happens when the connection is accepted for a different endpoint than the one receiving the connection request.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTCountDataBytes

Returns the amount of data currently available to be read. (Deprecated in Mac OS X v10.4.)

```
OTResult OTCountDataBytes (
   EndpointRef ref,
   OTByteCount *countPtr
);
```

Parameters
ref
countPtr

Return Value
See the description of the `OTResult` data type.

Discussion
If the function returns sucessfully, the countPtr parameter points to a buffer containing the amount of data currently available to be read. This does not mean that the buffer contains all the data that was sent. That is, there might be additional data to read after you do the first read.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTCreateConfiguration

Creates a structure defining a provider's configuration. (Deprecated in Mac OS X v10.4.)

Modified

```
OTConfigurationRef OTCreateConfiguration (
   const char *path
);
```

Parameters

path

A pointer to a character string describing the provider.

Return Value

See the description of the `OTConfigurationRef` data type.

Discussion

The `OTCreateConfiguration` function creates a configuration structure that defines the software modules, hardware ports, and options that Open Transport is to use when you call a function to open a provider. This is a private structure, defined by the `OTConfiguration` data type . To create one, you use the `path` parameter to pass the `OTCreateConfiguration` function a string describing the provider service desired.

The simplest possible value of the `path` parameter is a single protocol module name of the highest-level protocol you want to use; for example, "tcp." If you do not specify a complete communications path, the Open Transport software uses default settings to construct the rest of the path. For example, if you specify "adsp" for the `path` parameter, Open Transport defaults to using the AppleTalk DataStream Protocol (ADSP) protocol module layered above the Datagram Delivery Protocol (DDP) protocol module and with LocalTalk on the default port, which is the printer port.

If you want to identify a particular port in the configuration string, you use the port name to do so (described in the section "About Port Information," beginning on page 6-5). More typically, however, you leave this value blank— for example, using a string with only "adsp" or "adsp, ddp," which configures the provider with whatever port is specified in the control panel.

To specify more than one protocol module, separate the module names with commas. You can also specify values for options by putting them in parentheses after the protocol name; for example, "adsp, ddp (Checksum=1)" specifies that ADSP is to run on top of DDP and that the checksum option is enabled.

If Open Transport cannot parse the list that you pass in the `path` parameter, the `OTCreateConfiguration` function returns `((OTConfiguration*)-1L)`. If there is insufficient memory to create an `OTConfiguration` structure, the `OTCreateConfiguration` function returns `NULL`.

The OTCreateConfiguration function returns a pointer to the configuration structure it creates. You pass this pointer as a parameter to the open-provider functions such as the OTOpenEndpoint or OTOpenMapper functions.

Availability

Modified in Carbon. Available in CarbonLib 1.0 and later when OpenTransport 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.

Carbon Porting Notes

Passing inline options to `OTCreateConfiguration`-for example, `OTCreateConfiguration("tcp(NoDelay=1)")`-is not supported on Mac OS X. Instead, you should explicitly set any options using the function `OTOptionManagement`.

Declared In

`OpenTransport.h`

OTCreateDeferredTaskInContext

Creates a reference to a task that can be scheduled to run at deferred task time. (Deprecated in Mac OS X v10.4.)

```
long OTCreateDeferredTaskInContext (
   OTProcessUPP upp,
   void *arg,
   OTClientContextPtr clientContext
);
```

Parameters

upp

arg

clientContext

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`OpenTransport.h`

OTCreatePortRef

Creates a port reference that describes a port's hardware characteristics. (Deprecated in Mac OS X v10.4.)

```
OTPortRef OTCreatePortRef (
   OTBusType busType,
   OTDeviceType devType,
   OTSlotNumber slot,
   UInt16 other
);
```

Parameters

busType

The type of bus to which the hardware port is connected; for example, a NuBus or PCI bus. See "The Port Reference" for possible values for this parameter.

devType

The type of hardware device connected to the port, such as LocalTalk or Ethernet. See "The Port Reference"for possible values for this parameter.

slot
other

The port's multiport identifier—that is, a numeric value that distinguishes between ports when more than one hardware port is connected to a given slot.

Return Value
See the description of the `OTPortRef` data type.

Discussion
The `OTCreatePortRef` function creates a port reference structure, which is a 32-bit value that describes a port's hardware characteristics: its device and bus type, its physical slot number, and, where applicable, its multiport identifier.

Once you have created a port reference, you can use the `OTFindPortByRef` function to find a specific port with that particular set of characteristics.

To create a port reference, you use the `OTCreatePortRef` function. You must know all the port's hardware characteristics: its device and bus type, its slot number, and its multiport identifier (if it has one). You cannot use wildcards to fill in any element you don't know, although you can use a device type of 0 to allow matches on every kind of device type (following the zero-matches-everything rule). Possible device and bus types are described in the section "The Port Reference."

To create a port reference for a pseudodevice, use 0 as the value for the bus type, slot number, and multiport identifier, and use the constant `kOTPseudoDevice` for the device type.

Open Transport has predefined variants of the OTCreatePortRef function for the most commonly used hardware devices, such as the NuBus, PCI, and PCMCIA devices. These three variants are listed here:

```
#define OTCreateNuBusPortRef(devType, slot, other)\
OTCreatePortRef(kOTNuBus, devType, slot, other)
#define OTCreatePCIPortRef(devType, slot, other)\
OTCreatePortRef(kOTPCIBus, devType, slot, other)
#define OTCreatePCMCIAPortRef(devType, slot, other)\
OTCreatePortRef(kOTPCMCIABus, devType, slot, other)
```

Once you have identified the port structure you want, you can access the information in its port reference, by using the `OTGetDeviceTypeFromPortRef`, `OTGetBusTypeFromPortRef`, and `OTGetSlotFromPortRef` functions.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTCreateTimerTaskInContext

Creates a task to be scheduled. (Deprecated in Mac OS X v10.4.)

```
long OTCreateTimerTaskInContext (
   OTProcessUPP upp,
   void *arg,
   OTClientContextPtr clientContext
);
```

Parameters

upp

arg

clientContext

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProtocol.h`

OTDelay

Delays processing for a specified number of seconds. This function is only provided for compatibility with the UNIX `sleep` function. (Deprecated in Mac OS X v10.4.)

```
void OTDelay (
   UInt32 seconds
);
```

Parameters

seconds
The number of seconds to delay.

Discussion
The `OTDelay` function delays processing for the number of seconds specified in the `seconds` parameter. While the delay is occurring, `OTDelay` continuously calls the `OTIdle` function.

You can only call the `OTDelay` function from within an application at system task time. This function is only provided for compatibility with the UNIX `sleep` function to assist with portability of UNIX code.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTDeleteName

Removes a previously registered entity name. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTDeleteName (
   MapperRef ref,
   TNetbuf *name
);
```

Parameters

ref

name

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
If the name-registration protocol defined using the config parameter to the OTOpenMapper or OTAsyncOpenMapper function supports dynamic name and address registration, you can use the OTDeleteName function to delete a registered name.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTDeleteNameByID

Removes a previously registered name as specified by its name ID. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTDeleteNameByID (
   MapperRef ref,
   OTNameID nameID
);
```

Parameters

ref

nameID

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
If the name-registration protocol defined using the config parameter to the OTOpenMapper or OTAsyncOpenMapper function supports dynamic name and address registration, you can use the OTDeleteNameByID function to delete a registered name.

If the mapper is in asynchronous mode, the OTDeleteNameByID function returns immediately. When the function completes execution, the mapper provider calls the notifier function, passing T_DELNAMECOMPLETE for the code parameter, and a pointer to the id parameter in the cookie parameter.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTDequeue

Removes an element from a list. (Deprecated in Mac OS X v10.4.)

```
void * OTDequeue (
   void **listHead,
   OTByteCount linkOffset
);
```

Parameters

listHead

linkOffset

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTDestroyConfiguration

Deletes an `OTConfiguration` structure. (Deprecated in Mac OS X v10.4.)

```
void OTDestroyConfiguration (
   OTConfigurationRef cfig
);
```

Parameters

cfig

Discussion
The `OTDestroyConfiguration` function deletes the `OTConfiguration` structure that you specify in the cfig parameter and releases all associated memory.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTDestroyDeferredTask

Destroys a deferred task created with the `OTCreateDeferredTask` function. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTDestroyDeferredTask (
   OTDeferredTaskRef dtCookie
);
```

Parameters

dtCookie

Return Value

A result code. See "Open Transport Result Codes" (page 2722).

Discussion

The `OTDestroyDeferredTask` function makes the `dtCookie` reference invalid and frees any resources allocated to the task when it was created. You can call this function at any time when you no longer need to schedule the deferred task object. If `dtCookie` is invalid (a value of 0), the function returns `kOTNoError` and does nothing.

If you try to destroy a deferred task that is still scheduled, the `kEAgainErr` error can occur. This is a rare situation that can only happen when you try to destroy the task from within an interrupt service routine or within another deferred task.

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTDestroyTimerTask

Disposes of a timer task. (Deprecated in Mac OS X v10.4.)

```
void OTDestroyTimerTask (
   OTTimerTask timerTask
);
```

Parameters

timerTask

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`OpenTransportProtocol.h`

OTDontAckSends

Specifies that a provider copy data before sending it. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTDontAckSends (
   ProviderRef ref
);
```

Parameters

ref

Return Value

A result code. See "Open Transport Result Codes" (page 2722).

Discussion

By default, providers do not acknowledge sends. You need to call the `OTDontAckSends` function only if you have used the `OTAckSends` function to turn on send-acknowledgment for a provider.

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTElapsedMicroseconds

Calculates the time elapsed in microseconds since a specified time. (Deprecated in Mac OS X v10.4.)

```
UInt32 OTElapsedMicroseconds (
   OTTimeStamp *startTime
);
```

Parameters

startTime

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTElapsedMilliseconds

Calculates the time elapsed in milliseconds since a specified time. (Deprecated in Mac OS X v10.4.)

```
UInt32 OTElapsedMilliseconds (
   OTTimeStamp *startTime
);
```

Parameters

startTime

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

OpenTransport.h

OTEnqueue

Adds an element to a list. (Deprecated in Mac OS X v10.4.)

```
void OTEnqueue (
   void **listHead,
   void *object,
   OTByteCount linkOffset
);
```

Parameters

listHead

object

linkOffset

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

OpenTransport.h

OTEnterNotifier

Limits the notifications that can be sent to your notifier. (Deprecated in Mac OS X v10.4.)

```
Boolean OTEnterNotifier (
   ProviderRef ref
);
```

Parameters

ref

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTExtractNBPName

Extracts the name part of an NBP name from an NBP entity structure. (Deprecated in Mac OS X v10.4.)

```
void OTExtractNBPName (
   const NBPEntity *entity,
   char *name
);
```

Parameters

entity

name

A pointer to the string buffer in which to store the name portion of an NBP name string that you wish to extract from the NBP entity.

Discussion
The `OTExtractNBPName` function extracts the name part of an NBP name from the specified NBP entity structure and stores it into the string buffer specified by the `name` parameter. This function inserts a backslash (\) in front of any backslash, colon (:), or at-sign (@) it finds in an NBP name so that mapper functions can use a correctly formatted NBP name.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTExtractNBPType

Extracts the type part of an NBP name from an NBP entity structure. (Deprecated in Mac OS X v10.4.)

```
void OTExtractNBPType (
   const NBPEntity *entity,
   char *typeVal
);
```

Parameters

entity

typeVal

A pointer to the string buffer in which to store the type portion of an NBP name string that you wish to extract from the NBP entity.

Discussion
The `OTExtractNBPType` function extracts the type part of an NBP name from the specified NBP entity structure and stores it into the string buffer specified by the `type` parameter. This function inserts a backslash (\) in front of any backslash, colon (:), or at-sign (@) it finds in an NBP name so that mapper functions can use a correctly formatted NBP name.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTExtractNBPZone

Extracts the zone part of an NBP name from an NBP entity structure. (Deprecated in Mac OS X v10.4.)

```
void OTExtractNBPZone (
   const NBPEntity *entity,
   char *zone
);
```

Parameters
entity
zone
A pointer to the string buffer in which to store the type portion of an NBP name string that you wish to extract from the NBP entity.

Discussion
The `OTExtractNBPZone` function extracts the zone part of an NBP name from the specified NBP entity structure and stores it into the string buffer specified by the `zone` parameter. This function inserts a backslash (\) in front of any backslash, colon (:), or at-sign (@) it finds in an NBP name so that mapper functions can use a correctly formatted NBP name.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTFindAndRemoveLink

Finds a link in a FIFO list and removes it. (Deprecated in Mac OS X v10.4.)

```
OTLink * OTFindAndRemoveLink (
   OTList *list,
   OTListSearchUPP proc,
   const void *ref
);
```

Parameters

list

proc

ref

Return Value
See the description of the `OTLink` data type.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTFindLink

Finds a link in a FIFO list and returns a pointer to it. (Deprecated in Mac OS X v10.4.)

```
OTLink * OTFindLink (
   OTList *list,
   OTListSearchUPP proc,
   const void *ref
);
```

Parameters

list

proc

ref

Return Value
See the description of the `OTLink` data type.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTFindOption

Finds a specific option in an options buffer. (Deprecated in Mac OS X v10.4.)

```
TOption * OTFindOption (
   UInt8 *buffer,
   UInt32 buflen,
   OTXTILevel level,
   OTXTIName name
);
```

Parameters

buffer

A pointer to the buffer containing the option to be found.

buflen

The size of the buffer containing the option to be found.

level

name

Return Value

See the description of the `TOption` data type.

Discussion

Given a buffer such as might be returned by the `OTOptionManagement` function or by any endpoint function that returns a buffer containing option information, you can use the `OTFindOption` function to find a specific option in the buffer.

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTFindPort

Obtains information about a port that corresponds to a given port name. (Deprecated in Mac OS X v10.4.)

```
Boolean OTFindPort (
   OTPortRecord *portRecord,
   const char *portName
);
```

Parameters

portName

A pointer to a port structure that contains information about the port you specified with the `portName` parameter.

portName

The name of the port about which you want information.

Discussion

The `OTFindPort` function returns information about a port that corresponds to a given port name. Each port in a system has a unique port name, which you can obtain through a previous call or set of calls to the `OTGetIndexedPort` function.

You must allocate the port structure; the function fills this structure with information about the port indicated by the `portName` parameter. If the function returns `false`, the contents of the structure are not significant.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTFindPortByRef

Obtains information about a port that corresponds to its given port reference. (Deprecated in Mac OS X v10.4.)

```
Boolean OTFindPortByRef (
   OTPortRecord *portRecord,
   OTPortRef ref
);
```

Parameters
portRecord
ref

Discussion
The `OTFindPortByRef` function returns information about a port identified by its port reference. A port reference is a 32-bit value that describes a port's hardware characteristics: its bus and device type, its physical slot number, and, where applicable, its multiport identifier. This identifier differentiates between multiple hardware ports on a given slot.

You must allocate the port structure; the function fills this structure with information about the port indicated by the `ref` parameter. If the function returns `false`, the contents of the structure are not significant.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTFree

Frees memory allocated using the OTAlloc function. (Deprecated in Mac OS X v10.4.)

```
OTResult OTFree (
   void *ptr,
   OTStructType structType
);
```

Parameters

ptr

A pointer to the structure to be deallocated. This is the pointer returned by the OTAlloc function.

structType

Return Value
See the description of the `OTResult` data type.

Discussion
In order to use the OTFree function, you must not have changed the memory allocated by the OTAlloc function for the structure specified by the structType parameter or for any of the buffers to which it points.

You are responsible for passing a structType parameter that exactly matches the type of structure being freed.

The OTFree function, along with the OTAlloc function, is provided mainly forcompatibility with XTI.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTFreeMem

Frees memory allocated with the `OTAllocMem` function. (Deprecated in Mac OS X v10.4.)

```
void OTFreeMem (
   void *mem
);
```

Parameters

mem

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTGetBusTypeFromPortRef

Extracts the value of the bus type from a port reference. (Deprecated in Mac OS X v10.4.)

```
UInt16 OTGetBusTypeFromPortRef (
   OTPortRef ref
);
```

Parameters

ref

Discussion

The `OTGetBusTypeFromPortRef` function extracts the bus type value from a port reference with unknown hardware values. You can obtain such a port reference when another application passes one to you or when you use the `OTGetIndexedPort` function to access a port structure into which another application has put its own port reference.

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTGetClockTimeInSecs

Returns the number of seconds that have elapsed since system boot time. (Deprecated in Mac OS X v10.4.)

```
UInt32 OTGetClockTimeInSecs (
   void
);
```

Parameters

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTGetDeviceTypeFromPortRef

Extracts the value of the hardware device type from a port reference. (Deprecated in Mac OS X v10.4.)

```
OTDeviceType OTGetDeviceTypeFromPortRef (
   OTPortRef ref
);
```

Parameters

ref

Return Value
See the description of the `OTDeviceType` data type.

Discussion
The `OTGetDeviceTypeFromPortRef` function extracts the device type value from a port reference with unknown hardware values. You can obtain such a port reference when another application passes one to you or when you use the `OTGetIndexedPort` function to access a port structure into which another application has put its own port reference.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTGetEndpointInfo

Obtains information about an endpoint that has been opened. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTGetEndpointInfo (
   EndpointRef ref,
   TEndpointInfo *info
);
```

Parameters

ref

info

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
The OTGetEndpointInfo function returns information about

- the maximum size of buffers used to specify an endpoint's address and option values
- the maximum size of normal and expedited data you can transfer using this endpoint or, for transaction-based endpoints, the maximum size of requests and replies
- the size of data you can transfer when initiating or tearing down a connection
- the services supported by the endpoint
- any additional characteristics of this endpoint

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTGetEndpointState

Obtains the current state of an endpoint. (Deprecated in Mac OS X v10.4.)

```
OTResult OTGetEndpointState (
   EndpointRef ref
);
```

Parameters

ref

Return Value
See the description of the `OTResult` data type.

Discussion
The OTGetEndpointState function returns an integer greater than or equal to 0 indicating the state of the specified endpoint. The endpoint state enumeration describes possible endpoint states and lists their decimal value.

If the function fails, it returns a negative integer specifying the error code. You must open an endpoint before you can determine its state.

You might need to know an endpoint's state in order to determine whether a function has completed or whether the endpoint is in an appropriate state for the function that you want to call next.

This function returns endpoint state information immediately, whether the endpoint is in synchronous or asynchronous mode.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTGetFirst

Returns a pointer to the first element in a FIFO list. (Deprecated in Mac OS X v10.4.)

```
OTLink * OTGetFirst (
   OTList *list
);
```

Parameters

list

Return Value
See the description of the `OTLink` data type.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTGetIndexedLink

Returns a pointer to the link at a specified position in a FIFO list. (Deprecated in Mac OS X v10.4.)

```
OTLink * OTGetIndexedLink (
   OTList *list,
   OTItemCount index
);
```

Parameters

list

index

Return Value
See the description of the `OTLink` data type.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTGetIndexedPort

Iterates through the ports available on your computer. (Deprecated in Mac OS X v10.4.)

```
Boolean OTGetIndexedPort (
   OTPortRecord *portRecord,
   OTItemCount index
);
```

Parameters

portRecord

index

Discussion

The `OTGetIndexedPort` function returns information about the ports available on your local system. To iterate through all the ports on your computer, call the function repeatedly, incrementing the `index` parameter each time (starting with 0) until the function returns false. Each time the function returns true, it fills in the port structure that you provide with information about a specific port. You can use this information, for example, when specifying a provider configuration string for the `OTCreateConfiguration` function.

You must allocate the port structure; the function fills this structure with information about the port indicated by the `index` parameter. If the function returns `false`, the contents of the structure are not significant.

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTGetLast

Returns the last element in a FIFO list. (Deprecated in Mac OS X v10.4.)

```
OTLink * OTGetLast (
   OTList *list
);
```

Parameters

list

Return Value

See the description of the `OTLink` data type.

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTGetNBPEntityLengthAsAddress

Obtains the size of an NBP entity structure. (Deprecated in Mac OS X v10.4.)

```
OTByteCount OTGetNBPEntityLengthAsAddress (
   const NBPEntity *entity
);
```

Parameters

entity

Return Value

See the description of the `OTByteCount` data type.

Discussion

The `OTGetNBPEntityLengthAsAddress` function obtains the number of bytes needed to store an NBP entity structure into an NBP or combined DDP-NBP address structure.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`OpenTransportProviders.h`

OTGetProtAddress

Obtains the address to which an endpoint is bound and, if the endpoint is currently connected, obtains the address of its peer. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTGetProtAddress (
   EndpointRef ref,
   TBind *boundAddr,
   TBind *peerAddr
);
```

Parameters

ref

boundAddr

If you are calling this function only to determine the address of the peer endpoint, you can set the boundAddr parameter to NIL.

The boundAddr->qlen field is ignored. See `EndpointRef` data type.

peerAddr

Return Value

A result code. See "Open Transport Result Codes" (page 2722).

Discussion

The OTGetProtAddress function returns the address to which an endpoint is bound in the boundAddr parameter and, if the endpoint is currently connected, the address of its peer in the peerAddr parameter. Not all endpoints support this function. A value of T_XPG4_1 in the flags field of the `TEndpointInfo` (page 2542) structure indicates that the endpoint does support this function.

You are responsible for initializing the buffers required to hold the local and peer addresses. The addr field of the `TEndpointInfo` structure specifies the maximum amount of memory needed to store the address of an endpoint. Use this value to set the size of the buffers.

The information returned by the OTGetProtAddress function is affected by the state of the endpoint specified by the ref parameter. If the endpoint is in the T_UNBND state, the boundAddr->addr.len field is set to 0. If the endpoint is not in the T_DATAXFER state, the peerAddr->addr.len field is set to 0.

If the endpoint is in asynchronous mode and a notifier is not installed, it is not possible to determine when the function completes.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTGetSlotFromPortRef

Extracts slot information from a port reference. (Deprecated in Mac OS X v10.4.)

```
OTSlotNumber OTGetSlotFromPortRef (
   OTPortRef ref,
   UInt16 *other
);
```

Parameters

ref

other

A pointer to a 16-bit buffer you provide into which the function places a value that distinguishes between ports when more than one hardware port is connected to a given slot. Specify `NULL` for this parameter if you do not want the function to return this information.

Return Value
See the description of the `OTSlotNumber` data type.

Discussion
The `OTGetSlotFromPortRef` function extracts slot information from a port reference with unknown hardware values. You can obtain such a port reference when another application passes one to you or when you use the `OTGetIndexedPort` function to access a port structure into which another application has put its own port reference.

Note that the slot numbers are physical; that is, they are the slot numbers returned by the Slot Manager and not the slots seen in various network configuration applications. Physical slot numbers depend on the type of card installed. For example, NuBus cards number their slots 9–13, which appear in the AppleTalk or TCP control panels as slots 1–5.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTGetTimeStamp

Obtains the current timestamp. (Deprecated in Mac OS X v10.4.)

```
void OTGetTimeStamp (
   OTTimeStamp *currentTime
);
```

Parameters

currentTime

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTIdle

Idles your computer. (Deprecated in Mac OS X v10.4.)

```
void OTIdle (
   void
);
```

Discussion
You can call the `OTIdle` function while you are waiting for asynchronous provider operations to complete. It is not necessary for the correct operation of Open Transport to call this function, but it provides compatibility for existing programs that use an idling function.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTInetAddressToName

Determines the canonical domain name of the host associated with an internet address. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTInetAddressToName (
   InetSvcRef ref,
   InetHost addr,
   InetDomainName name
);
```

Parameters

ref

addr

name

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
If you call this function asynchronously, the TCP/IP service provider calls your notifier function with the `T_DNRADDRTONAMECOMPLETE` completion event code when the function completes. The `cookie` parameter to the notifier function contains a pointer to the `InetHost` structure you specified in the `addr` parameter. If you had more than one simultaneous outstanding call to the `OTInetAddressToName` function, you can use this information to determine which call has completed execution.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTInetGetInterfaceInfo

Returns internet address information about the local host. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTInetGetInterfaceInfo (
   InetInterfaceInfo *info,
   SInt32 val
);
```

Parameters

info

val

An index into the local host's array of configured IP interfaces. Specify 0 for information about the first interface. Specify `kDefaultInetInterface` to get information about the primary interface.

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
Because the architecture of Open Transport TCP/IP provides for multihoming, in principle a given host can receive packets simultaneously through more than one network interface. For each IP interface configured for the local host, the `OTInetGetInterfaceInfo` function provides the internet address and subnet mask, a default gateway (that is, a gateway, if any exists, that can be used to route any packet to all destinations outside the locally connected subnet), and a domain name server, if any is known. The function also returns

the version number of the `OTInetGetInterfaceInfo` function and, if available, the broadcast address for each interface. If the broadcast address is not available, you can determine it from the internet address and subnet mask.

Because multihoming has not been implemented in the initial release of Open Transport, the `OTInetGetInterfaceInfo` function never returns information for more than one interface.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTInetGetSecondaryAddresses

Returns the active secondary IP addresses. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTInetGetSecondaryAddresses (
   InetHost *addr,
   UInt32 *count,
   SInt32 val
);
```

Parameters
addr
count
val

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTInetHostToString

Converts an an address in InetHost format into a character string in dotted-decimal notation. (Deprecated in Mac OS X v10.4.)

```
void OTInetHostToString (
   InetHost host,
   char *str
);
```

Parameters

host

str

A pointer to a C string containing an IP address in dotteddecimal notation (for example, “12.13.14.15”). You must allocate storage for this string and provide the pointer to the function.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`OpenTransportProviders.h`

OTInetMailExchange

Returns mail-exchange-host names and preference information for a domain name you specify. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTInetMailExchange (
   InetSvcRef ref,
   char *name,
   UInt16 *num,
   InetMailExchange *mx
);
```

Parameters

ref

name

A pointer to a host name, partially qualified domain name, or fully qualified domain name for which you want mail exchange information.

num

A pointer to the number of elements in the array pointed to by the `mx` parameter. When the function completes, it sets the number pointed to by the `num` parameter to the actual number of elements filled in.

mx

Return Value

A result code. See “Open Transport Result Codes” (page 2722).

Discussion

In order to deliver mail, a mail application must determine the fully qualified domain name of the host to which the mail should be sent. That host might be the final destination of the mail, a mail server, or a router. The domain name system servers maintain mail-exchange resource records that pair domain names with the hosts that can accept mail for that domain. Each domain name can be paired with any number of host

names; each record containing such a pair also contains a preference number. The mailer sends the mail to the host with the lowest preference number first and tries the others in turn until the mail is delivered or until the mailer decides that the mail is undeliverable.

The `OTInetMailExchange` function returns mail-exchange-host and preference information for the domain name you specify. You must then determine the address of the host and how best to deliver the mail. You can specify as many elements to the array of `InetMailExchange` structures as you wish.

If you call this function asynchronously, the TCP/IP service provider calls your notifier function with the `T_DNRMAILEXCHANGECOMPLETE` completion event code when the function completes. The `cookie` parameter to the notifier function contains the array pointer you specified in the `mx` parameter. If you had more than one simultaneous outstanding call to the `OTInetMailExchange` function, you can use this information to determine which call has completed execution.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTInetQuery

Executes a generic DNS query. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTInetQuery (
   InetSvcRef ref,
   char *name,
   UInt16 qClass,
   UInt16 qType,
   char *buf,
   OTByteCount buflen,
   void **argv,
   OTByteCount argvlen,
   OTFlags flags
);
```

Parameters
ref
name
qClass
qType
buf
buflen
argv
argvlen
flags

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTInetStringToAddress

Resolves a domain name to its equivalent internet addresses. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTInetStringToAddress (
   InetSvcRef ref,
   char *name,
   InetHostInfo *hinfo
);
```

Parameters

ref

name

A pointer to the domain name you want to resolve. This can be a host name, a partially qualified domain name, a fully qualified domain name, or an internet address in dotted-decimal format.

hinfo

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
Because the architecture of Open Transport TCP/IP provides for multihoming, a single host can be associated with multiple internet addresses. You can use the `OTInetStringToAddress` function to return multiple addresses for multihomed hosts.

Because multihoming has not been implemented in the initial release of Open Transport, the `OTInetStringToAddress` function never returns more than one address.

If you specify an internet address in dotted-decimal format for the `hinfo` parameter, the `OTInetStringToAddress` function places that address in the `InetHostInfo.name` field instead of a canonical domain name.

If you call the `OTInetStringToAddress` function asynchronously, the TCP/IP service provider calls your notifier function with the `T_DNRSTRINGTOADDRCOMPLETE` completion event code when the function completes. The `cookie` parameter to the notifier function contains the pointer you specified in the `hinfo` parameter. If you had more than one simultaneous outstanding call to the `OTInetStringToAddress` function, you can use this information to determine which call has completed execution.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTInetStringToHost

Converts an IP address string from dotted-decimal notation or hexadecimal notation to an `InetHost` data type. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTInetStringToHost (
   const char *str,
   InetHost *host
);
```

Parameters

str

A pointer to a character string containing an IP address in either dotted-decimal notation (for example, "12.13.14.15") or hexadecimal notation (for example, "0x0c0d0e0f").

host

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTInetSysInfo

Returns details about a host's processor and operating system. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTInetSysInfo (
   InetSvcRef ref,
   char *name,
   InetSysInfo *sysinfo
);
```

Parameters

ref

name

The name of the host about which you want information. This can be a host name (including the local host), a partially qualified domain name, or a fully qualified domain name.

sysinfo

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
The information returned by this function is maintained by the domain name server. If you call this function asynchronously, the TCP/IP service provider calls your notifier function with the `T_DNRSYSINFOCOMPLETE` completion event code when the function completes. The `cookie` parameter to the notifier function contains a pointer to the `InetSysInfo` structure you specified in the `sysinfo` parameter. If you had more than one simultaneous outstanding call to the `OTInetSysInfo` function, you can use this information to determine which call has completed execution.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTInitDDPAddress

Initializes a DDP address structure. (Deprecated in Mac OS X v10.4.)

```
void OTInitDDPAddress (
   DDPAddress *addr,
   UInt16 net,
   UInt8 node,
   UInt8 socket,
   UInt8 ddpType
);
```

Parameters

addr

net
The network number you wish to specify. Set to 0 to default to the local network.

node
The node ID you wish to specify. Set to 0 to default to the local node.

socket
The socket number you wish to specify. Set to 0 to allow Open Transport to assign a socket dynamically when you use this address to bind an endpoint.

ddpType
The DDP type you wish to specify. Set to 0 unless you are using DDP.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTInitDDPNBPAddress

Initializes a combined DDP-NBP address structure. (Deprecated in Mac OS X v10.4.)

```
OTByteCount OTInitDDPNBPAddress (
   DDPNBPAddress *addr,
   const char *name,
   UInt16 net,
   UInt8 node,
   UInt8 socket,
   UInt8 ddpType
);
```

Parameters

addr

name

A pointer to the NBP string you wish to use for the NBP name.

net

The network number you wish to specify. Set to 0 to default to the local network.

node

The node ID you wish to specify. Set to 0 to default to the local node.

socket

The socket number you wish to specify. Set to 0 to allow Open Transport to assign a socket dynamically when you use this address to bind an endpoint.

ddpType

The DDP type you wish to specify. Set to 0 unless you are using DDP.

Return Value

See the description of the `OTByteCount` data type.

Discussion

The `OTInitDDPNBPAddress` function initializes a combined DDP-NBP address structure with the data provided in the parameters: NBP name, network number, node ID, socket number, and DDP type. The function returns the total size of the address structure, which is the length of the `name` parameter plus the size of a `DDPAddress` structure.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`OpenTransportProviders.h`

OTInitDNSAddress

Fills in a DNSAddress structure with the data you provide. (Deprecated in Mac OS X v10.4.)

```
OTByteCount OTInitDNSAddress (
   DNSAddress *addr,
   char *str
);
```

Parameters

addr

str

A pointer to a domain name string. This string can be just a host name (otteam), a partially qualified domain name (for example, "otteam.ssw"), a fully qualified domain name (for example, "otteam.ssw.apple.com."), or an internet address in dotteddecimal format (for example, "17.202.99.99"), and can optionally include the port number (for example, "otteam.ssw.apple.com:25" or "17.202.99.99:25").

Return Value
See the description of the `OTByteCount` data type.

Discussion
This function fills in the `fAddressType` field of the `DNSAddress` structure with the value `AF_DNS`, fills in the `fName` field with the address string you specify, and returns the size of the resulting `DNSAddress` structure as an unsigned integer. You can use the `DNSAddress` structure to provide an address when you use a UDP or TCP endpoint. If you do so, the domain name resolver resolves the address for you automatically.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTInitInetAddress

Fills in an InetAddress structure with the data you provide. (Deprecated in Mac OS X v10.4.)

```
void OTInitInetAddress (
   InetAddress *addr,
   InetPort port,
   InetHost host
);
```

Parameters

addr

port

host

Discussion
This function fills in the `fAddressType` field of the `InetAddress` structure with the value `AF_INET`. You use the InetAddress structure when providing a TCP or UDP address to the Open Transport functions `OTConnect`, `OTSndURequest`, and `OTBind`. You are not required to use the `OTInitInetAddress` function when creating an `InetAddress` structure; this function is provided for your convenience only.

Availability
Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTInitNBPAddress

Initializes an NBP address structure. (Deprecated in Mac OS X v10.4.)

```
OTByteCount OTInitNBPAddress (
   NBPAddress *addr,
   const char *name
);
```

Parameters

addr

name

A pointer to the NBP string you wish to use for the NBP name.

Return Value
See the description of the `OTByteCount` data type.

Discussion
The `OTInitNBPAddress` function can be used to initialize an NBP address structure with the NBP name specified in the `name` parameter, which is assumed to already be in the correct string format. The function returns the size of the NBP address structure, which is the size of the `fAddressType` field plus the length of the string in the `name` parameter.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTInitNBPEntity

Initializes an NBP entity structure. (Deprecated in Mac OS X v10.4.)

```
void OTInitNBPEntity (
   NBPEntity *entity
);
```

Parameters

entity

Discussion
The `OTInitNBPEntity` function initializes an NBP entity structure, setting the name, type and zone parts of an NBP name to empty strings.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTInstallNotifier

Installs a notifier function. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTInstallNotifier (
   ProviderRef ref,
   OTNotifyUPP proc,
   void *contextPtr
);
```

Parameters

ref

proc

For C++ applications, the `proc` parameter must point to either a C function or a static member function. See `OTNotifyUPP` data type.

contextPtr

A context pointer for your use. The provider passes this value unchanged to your notifier function when it calls the function.

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
The `OTInstallNotifier` function installs a notifier function for the provider that you specify. Changing a provider's mode of execution does not affect the notifier function. The notifier function remains installed until you remove it using the `OTRemoveNotifier` function or until you close the provider.

Before calling the `OTInstallNotifier` function, you must open the provider for which you want to install the notifier. If you open a provider asynchronously (for example, with the `OTAsyncOpenEndpoint` function), you must pass a pointer to a notifier function as a parameter to the function used to open the provider. In this case, you don't need to call the `OTInstallNotifier` function unless you want to install a different notifier function. If you do, you must call the `OTRemoveNotifier` function before calling the `OTInstallNotifier` function.

Opening a provider synchronously (for example, with the `OTOpenEndpoint` function) opens the provider but does not install a notifier function for it. If you need a notifier function for a provider opened synchronously, you must call the `OTInstallNotifier` function. This notifier would not return completion events, but would return asynchronous events advising you of the arrival of data, of changes in flow-control restrictions, and so on.

Call the `OTInstallNotifier` function only when no provider functions are executing for the provider that you specify. Otherwise, the `OTInstallNotifier` function returns the result code `kOTStateChangeErr`.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTIoctl

Sends a module-specific command to an Open Transport protocol module. (Deprecated in Mac OS X v10.4.)

```
SInt32 OTIoctl (
   ProviderRef ref,
   UInt32 cmd,
   void *data
);
```

Parameters

ref

cmd

A routine selector for the module-specific command.

data

Data to be used by the module-specific command, or a pointer to such data. The interpretation of the `data` parameter is command specific.

Discussion
The `OTIoctl` function sends a module-specific command to an Open Transport protocol module. The `OTIoctl` function runs synchronously or asynchronously, matching the provider's mode of execution.

If the `OTIoctl` function completes synchronously without error, it returns 0 or a positive integer. The positive integer's meaning is command specific. If the `OTIoctl` function fails while executing synchronously, its return value is a negative integer corresponding to an Open Transport result code.

If the `OTIoctl` function runs asynchronously, it returns immediately with a return value `kOTNoError` or another Open Transport result code. When the function completes execution, Open Transport calls the notifier function you specify, passing the event code `kStreamIoctlEvent` and a `result` parameter indicating the result of the completed `OTIoctl` function. If the value of the `result` parameter is greater than 0, the corresponding result code is defined by the command; otherwise, the value of the `result` parameter corresponds to an Open Transport result code.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTIsAckingSends

Determines whether a provider is acknowledging sends. (Deprecated in Mac OS X v10.4.)

```
Boolean OTIsAckingSends (
   ProviderRef ref
);
```

Parameters

ref

Discussion

The `OTIsAckingSends` function returns `true` if the provider acknowledges sends and `false` if it does not.

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTIsBlocking

Returns a boolean indicating whether a provider is blocking. (Deprecated in Mac OS X v10.4.)

```
Boolean OTIsBlocking (
   ProviderRef ref
);
```

Parameters

ref

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTIsInList

Determines whether the specified link is in the specified list. (Deprecated in Mac OS X v10.4.)

```
Boolean OTIsInList (
   OTList *list,
   OTLink *link
);
```

Parameters

list

link

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTIsSynchronous

Returns a provider's current mode of execution. (Deprecated in Mac OS X v10.4.)

```
Boolean OTIsSynchronous (
   ProviderRef ref
);
```

Parameters

ref

Discussion

The `OTIsSynchronous` function returns `true` if a provider is in synchronous mode or returns `false` if the provider is in asynchronous mode.

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTLeaveNotifier

Allows Open Transport to resume sending primary and completion events. (Deprecated in Mac OS X v10.4.)

```
void OTLeaveNotifier (
   ProviderRef ref
);
```

Parameters

ref

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTLIFODequeue

Removes the first link in a LIFO list and returns a pointer to it. (Deprecated in Mac OS X v10.4.)

```
OTLink * OTLIFODequeue (
   OTLIFO *list
);
```

Parameters

list

Return Value
See the description of the `OTLink` data type.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTLIFOEnqueue

Places a link at the front of a LIFO list. (Deprecated in Mac OS X v10.4.)

```
void OTLIFOEnqueue (
   OTLIFO *list,
   OTLink *link
);
```

Parameters

list

link

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTLIFOStealList

Removes all links in a LIFO list and returns a pointer to the first link in the list. (Deprecated in Mac OS X v10.4.)

```
OTLink * OTLIFOStealList (
   OTLIFO *list
);
```

Parameters
list

Return Value
See the description of the `OTLink` data type.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTListen

Listens for an incoming connection request. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTListen (
   EndpointRef ref,
   TCall *call
);
```

Parameters
ref
call

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
You use the OTListen function to listen for incoming connection requests. On return, the function fills in the TCall structure referenced by the call parameter with information about the connection request. After retrieving the connection request using the OTListen function, you can reject the request using the OTSndDisconnect function, or you can accept the request using the OTAccept function.

If the endpoint is in synchronous mode and is blocking, the OTListen function returns when a connection request has arrived. If the endpoint is in asynchronous mode or is not blocking, the OTListen function returns any pending connection requests or returns the kOTNoDataErr result if there are no pending connection requests. You can also call the OTListen function from within a notifier function in response to the T_LISTEN event. In this case, the function returns a result immediately.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTLook

Determines the current asynchronous event pending for an endpoint. (Deprecated in Mac OS X v10.4.)

```
OTResult OTLook (
   EndpointRef ref
);
```

Parameters
ref

Return Value
See the description of the `OTResult` data type.

Discussion
You use the OTLook function in one of two cases. First, if the endpoint is in synchronous mode, you can call the OTLook function to poll for incoming data or connection requests. Second, certain asynchronous events might cause a synchronous function to fail with the result kOTLookErr. For example, if you call OTAccept and the endpoint gets a T_DISCONNECT event, the OTAccept function returns with kOTLookErr. In this case, you need to call the OTLook function to determine what event caused the original function to fail. Table 3-7 on page 3-26 lists the functions that might return the kOTLookErr result and the events that can cause these functions to fail.

The OTLook function returns an integer value that specifies the asynchronous event pending for the endpoint specified by the ref parameter. On error, OTLook returns a negative integer corresponding to a result code.

If there are multiple events pending, the OTLook function first looks for one ofthe following events: T_LISTEN, T_CONNECT, T_DISCONNECT, T_UDERR, or T_ORDREL. If it finds more than one of these, it returns them to you in first-in, first-out order. After processing these events, the OTLook function looks for the T_DATA, T_REQUEST, and T_REPLY events. If it finds more than one of these, it returns them to you in first-in, first-out order. You cannot use the OTLook function to poll for completion events.

Unless you are operating exclusively in synchronous mode, it is recommended that you use notifier functions to get information about pending events for an endpoint.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTLookupName

Finds and returns all addresses that correspond to a particular name or name pattern, or confirms that a name is registered. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTLookupName (
   MapperRef ref,
   TLookupRequest *req,
   TLookupReply *reply
);
```

Parameters

ref

req

reply

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
You can use the OTLookupName function to find out whether a name is registered and what address is associated with that name. You use the req parameter to supply the information needed for the search: what name should be looked up and, optionally, what node contains that information, how many matches you expect to find, and how long the search should continue before the function returns. On return, the reply parameter contains the names field that points to the buffer where the matching entries are stored and the rspcount field that specifies the number of matching entries.

For each registered name found, the OTLookupName function stores the following information in the buffer referenced by the names field of the reply parameter:

```
unsigned short addrLen; /* length of address that follows*/
unsigned short nameLen; /* length of name that follows */
unsigned char addr[]; /* address */
unsigned char name[]; /* name, padded to quad-word boundary*/
```

If you are searching for names using a name pattern and you expect that more than one name will be returned to you, you need to parse the reply buffer to extract the matching names.

If you call the OTLookupName function asynchronously, the mapper provider calls your notifier function passing one of two completion codes for the code parameter (T_LKUPNAMERESULT or T_LKUPNAMECOMPLETE) and passing the reply parameter in the cookie parameter. The mapper provider passes the T_LKUPNAMERESULT code each time it stores a name in the reply buffer, and it passes the T_LKUPNAMECOMPLETE code when it is done. When you receive this event, examine the rspcount field to determine whether there is a last name to retrieve from the reply buffer. The use of both codes is a feature that gives you a choice about how to process multiple names when searching for names matching a pattern.

- If you decide to allocate a buffer that is large enough to contain all the names returned, you can ignore the T_LKUPNAMERESULT code and call a function that parses the buffer once the OTLookupName function has completed—that is, once the provider calls your notifier function using the T_LKUPNAMECOMPLETE event.

- If you want to save memory or if you don't know how large a buffer to allocate, you can use the following method to process the names returned. Each time that the T_LKUPNAMERESULT event is passed, you must do something with the reply from the reply buffer. You can copy it somewhere, or you can delete it if it isn't a name you're interested in. Then, from inside your notifier you must set the reply->names.len field or the reply->rspcount field back to 0 (thus allowing the mapper provider to overwrite the original name). This tells the mapper provider that you are ready to receive another name. Accordingly, when the mapper provider has inserted another name into your reply buffer, it calls your notifier passing the T_LKUPNAMERESULT code, and you can process the new entry as you have processed the first entry. This method also saves you the trouble of having to parse through the buffer to extract name and address information.

The cookie parameter to the notifier contains the reply parameter.

The format of the names and protocol addresses are specific to the underlying protocol. Consult the documentation supplied for your protocol for more information.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTMemcmp

Compares the contents of two memory locations. (Deprecated in Mac OS X v10.4.)

```
Boolean OTMemcmp (
   const void *mem1,
   const void *mem2,
   OTByteCount nBytes
);
```

Parameters
mem1
mem2
nBytes

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTMemcpy

Copies data from one memory location to another; the source and destination locations must not overlap. (Deprecated in Mac OS X v10.4.)

```
void OTMemcpy (
   void *dest,
   const void *src,
   OTByteCount nBytes
);
```

Parameters

dest

src

nBytes

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

OpenTransport.h

OTMemmove

Copies data from one memory location to another; the source and destination locations may overlap. (Deprecated in Mac OS X v10.4.)

```
void OTMemmove (
   void *dest,
   const void *src,
   OTByteCount nBytes
);
```

Parameters

dest

src

nBytes

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

OpenTransport.h

OTMemset

Sets the specified memory range to a specific value. (Deprecated in Mac OS X v10.4.)

```
void OTMemset (
   void *dest,
   OTUInt8Param toSet,
   OTByteCount nBytes
);
```

Parameters

dest

toSet

nBytes

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTMemzero

Initializes the specified memory range to 0. (Deprecated in Mac OS X v10.4.)

```
void OTMemzero (
   void *dest,
   OTByteCount nBytes
);
```

Parameters

dest

nBytes

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTNextOption

Locates the next `TOption` structure in a buffer. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTNextOption (
   UInt8 *buffer,
   UInt32 buflen,
   TOption **prevOptPtr
);
```

Parameters

buffer

A pointer to the buffer containing the option to be found.

buflen

A long specifying the size of the buffer containing the option to be found.

prevOptPtr

Return Value

A result code. See "Open Transport Result Codes" (page 2722).

Discussion

The `OTNextOption` function allows you to parse through a buffer containing `TOption` structures describing an endpoint's option values. Within the buffer, `TOption` structures are aligned to long-word boundaries. This function takes into account this padding when it calculates the beginning address of the next `TOption` structure and it returns that address in the `prevOptPtr` parameter.

The first time you call the option, set the `prevOptPtr` parameter to the beginning address of the buffer. When the function returns, the `prevOptPtr` parameter points to the next (second) option in the buffer. You can continue this process, specifying the value returned for the `prevOptPtr` parameter by the previous invocation of the function, each time you call the function to obtain the beginning address of each option in the buffer.

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTOpenAppleTalkServicesInContext

Opens a synchronous AppleTalk service provider. (Deprecated in Mac OS X v10.4.)

```
ATSvcRef OTOpenAppleTalkServicesInContext (
   OTConfigurationRef cfig,
   OTOpenFlags flags,
   OSStatus *err,
   OTClientContextPtr clientContext
);
```

Parameters

cfig

flags

err

clientContext

Return Value
See the description of the `ATSvcRef` data type.

Discussion
Applications may pass a NULL context pointer but nonapplications must always pass a valid client context pointer.

You receive a client context pointer when you call the function `InitOpenTransportInContext` (page 2303).

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTOpenEndpointInContext

Opens an endpoint that operates synchronously. (Deprecated in Mac OS X v10.4.)

```
EndpointRef OTOpenEndpointInContext (
   OTConfigurationRef config,
   OTOpenFlags oflag,
   TEndpointInfo *info,
   OSStatus *err,
   OTClientContextPtr clientContext
);
```

Parameters

config

oflag

info

err

clientContext

Return Value
See the description of the `EndpointRef` data type.

Availability
Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransport.h`

OTOpenInternetServicesInContext

Opens the TCP/IP service provider and returns an internet services reference. (Deprecated in Mac OS X v10.4.)

```
InetSvcRef OTOpenInternetServicesInContext (
   OTConfigurationRef cfig,
   OTOpenFlags oflag,
   OSStatus *err,
   OTClientContextPtr clientContext
);
```

Parameters
cfig
oflag
err
clientContext

Return Value
See the description of the `InetSvcRef` data type.

Discussion
Applications may pass a NULL context pointer but nonapplications must always pass a valid client context pointer.

You receive a client context pointer when you call the function `InitOpenTransportInContext` (page 2303).

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTOpenMapperInContext

Creates a synchronous mapper provider and returns a mapper reference. (Deprecated in Mac OS X v10.4.)

```
MapperRef OTOpenMapperInContext (
   OTConfigurationRef config,
   OTOpenFlags oflag,
   OSStatus *err,
   OTClientContextPtr clientContext
);
```

Parameters

config

oflag

err

clientContext

Return Value
See the description of the `MapperRef` data type.

Discussion
Applications may pass a NULL pointer but non-applications must always pass a valid client context pointer.

You receive a client context pointer when you call the function `InitOpenTransportInContext` (page 2303).

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransport.h`

OTOptionManagement

Determines an endpoint's current or default option values or changes these values. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTOptionManagement (
   EndpointRef ref,
   TOptMgmt *req,
   TOptMgmt *ret
);
```

Parameters

ref

req

ret

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
To use the `OTOptionManagement` function, you must have opened an endpoint using the `OTOpenEndpoint` or `OTAsyncOpenEndpoint` functions.

You use the `OTOptionManagement` function to negotiate, retrieve, or verify an endpoint's protocol options. If the endpoint is in asynchronous mode and you have not installed a notifier function, it is not possible to determine when the function completes.

The action taken by the `OTOptionManagement` function is determined by the setting of the `req->flags` field. The following bulleted items describe the different operations that you can perform and the flag settings that you use to specify these operations.

- To negotiate values for the endpoint, you must call the `OTOptionManagement` function, specifying `T_NEGOTIATE` for the `req->flags` field. The endpoint provider evaluates the requested options, negotiates the values, and returns the resulting values in the option management structure pointed to by the `ret->opt.buf` field. The `status` field of each returned option is set to a constant that indicates the result of the negotiation. These constants are described by the "Open Transport Flags and Status Codes" (page 2702) enumeration.

  For any protocol specified, you can negotiate for the default values of all options supported by the endpoint by specifying the value `T_ALLOPT` for the `name` field of the `TOption` structure. This might be useful if you want to change current settings or if negotiations for other values have failed. The success of the negotiations depends partly on the state of the endpoint—that is, simply because these are default values does not guarantee a completely successful negotiation. When the function returns, the resulting values are returned, option by option, in the buffer pointed to by the `ret->opt.buf` field.

- To retrieve an endpoint's default option values, call the `OTOptionManagement` function, specifying `T_DEFAULT` for the `req->flags` field. You must also specify the name of the option (but not its value) in the `TOption` structure that you create for each of the options you are interested in.

  When the function returns, it passes the default values for the options back to you in the buffer pointed to by the `ret->opt.buf` field. For each option, the `status` field contains `T_NOTSUPPORT` if the protocol does not support the option, `T_READONLY` if the option is read-only, and `T_SUCCESS` in all other cases. The overall result of the request is returned in the `ret->flags` field. The meaning of this result is described by the `Open Transport Flags and Status Codes` enumeration.

  When getting an endpoint's default option values, you can specify `T_ALLOPT` for the option name. This returns all supported options for the specified level with their default values. In this case, you must set the `opt.maxlen` field to the maximum size required to hold an endpoint's option information. The `info.opt` field of the `TEndpointInfo` (page 2542) structure specifies the maximum size of a buffer used to hold option information for an endpoint.

- To retrieve an endpoint's current option values, call the `OTOptionManagement` function, specifying `T_CURRENT` for the `req->flags` field. For each option in the buffer referenced by the `req->opt.buf` field, specify the name of the option you are interested in. The function ignores any option valuesyou specify.

  When the function returns, it passes the current values for the options back to you in the buffer referenced by the `ret->opt.buf` field. For each option, the `status` field contains `T_NOTSUPPORT` if the protocol does not support the option, `T_READONLY` if the option is read-only, and `T_SUCCESS` in all other cases. The overall result of the request is returned in the `ret->flags` field. The meaning of this result is described by the "Open Transport Flags and Status Codes" (page 2702) enumeration.

  When retrieving an endpoint's current option values, you can specify `T_ALLOPT` for the option name. The function returns all supported options for the specified protocol, with their current values. In this case, you must set the `opt.maxlen` field to the maximum size required to hold an endpoint's option information. The `info.opt` field of the `TEndpointInfo` structure specifies the maximum size of a buffer used to hold option information for an endpoint.

- To check whether an endpoint provider supports certain options or option values, you must call the OTOptionManagement function, specifying T_CHECK for the req->flags field. Checking options or their values does not change the current settings of an endpoint's options.
  - To check whether an option is supported, set the `name` field of the `TOption` structure to the option name, but do not specify an option value. When the function returns, the `status` field for the corresponding `TOption` structure in the buffer pointed to by the `ret->opt.buf` field is set to `T_SUCCESS` if the option is supported, `T_NOTSUPPORT` if it is not supported or needs additional client privileges, and `T_READONLY` if it is read-only.
  - To check whether an option value is supported, set the `name` field of the `TOption` structure to the option name, and set the `value` field to the value you want to check. When the function returns, the `status` field for the corresponding `TOption` structure in the buffer pointed to by the `ret->opt.buf` field is set as it would be if you had specified the `T_NEGOTIATE` flag. The overall result of the option checks is returned in the `ret->flags` field, which contains the single worst result of the option checks. The meaning of this result is described by the `Open Transport Flags and Status Codes` enumeration.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTRcv

Reads data sent using a connection-oriented transactionless protocol. (Deprecated in Mac OS X v10.4.)

```
OTResult OTRcv (
   EndpointRef ref,
   void *buf,
   OTByteCount nbytes,
   OTFlags *flags
);
```

Parameters

ref

buf

A pointer to a memory location where the incoming data is to be copied. You must allocate this buffer before you call the function.

nbytes

flags

Return Value
See the description of the `OTResult` data type.

Discussion
You call the `OTRcv` function to read data sent by the peer to which you are connected. If the `OTRcv` function succeeds, it returns an integer (`OTStatus`) specifying the number of bytes received. The function places the data read into the buffer referenced by the buf parameter. If the function fails, it returns a negative integer corresponding to a result code that indicates the reason for the failure. You can call this function to receive either normal or expedited data. If the data is expedited, the `T_EXPEDITED` flag is set in the flags parameter.

If `T_MORE` is set in the flags parameter when the function returns, this means that the buffer you allocated is too small to contain the data to be read and that you must call the `OTRcv` function again. If you have read x bytes with the first call, the next call to the `OTRcv` function begins to read at the (x + 1) byte. Of course, if you need it, you must copy the data in the buffer to another location before calling the function again. Each call to this function that returns with the `T_MORE` flag set means that you must call the function again to get more data. When you have read all the data, the OTRcv function returns with the `T_MORE` flag not set. If the endpoint does not support the concept of a TSDU (Transport Service Data Unit), the `T_MORE` flag is not meaningful and should be ignored. To determine whether the endpoint supports TSDUs, examine the `tsdu` field of the `TEndpointInfo` (page 2542) structure. A value of `T_INVALID` means that the endpoint does not support it.

Some protocols allow you to send zero-length data to signal the end of a logical unit. In this case, if you request more than 0 bytes when calling the OTRcv function, the function returns 0 bytes only to signal the end of a TSDU.

If the OTRcv function returns and the T_EXPEDITED bit is set in the flags parameter, this means that you are about to read expedited data. If the number of bytes of expedited data exceeds the number of bytes you specified in the reqCount parameter, both the T_EXPEDITED and the T_MORE bits are set. You must call the OTRcv function until the T_MORE flag is not set to retrieve the rest of the expedited data.

If you are calling the OTRcv function repeatedly to read normal data and a call to the function returns T_EXPEDITED in the flags parameter, the next call to the OTRcv function that returns without the T_EXPEDITED flag set returns normal data at the place it was interrupted. It is your responsibility to remember where that was and to continue processing normal data. You can determine how much normal data you read by maintaining a running total of the number of bytes returned in the OTStatus result.

If the endpoint is in asynchronous mode or is not blocking, the function returns with the kOTNoDataErr result if no data is available. If you have installed a notifier, the endpoint provider calls your notifier and passes T_DATA or T_EXDATA for the code parameter when there is data available. If you have not installed a notifier, you may poll for these events using the OTLook function. Once you receive a T_DATA or T_EXDATA event, you should continue in a loop, calling the OTRcv function until it returns with the kOTNoDataErr result.

If the endpoint is in synchronous mode and is blocking, the endpoint waits for data if none is currently available. You should avoid calling the OTRcv function this way because it might cause processing to hang if no data is available. If you are doing other operations in synchronous mode, you should put the endpoint in nonblocking mode before calling the OTRcv function.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTRcvConnect

Reads the status of an outstanding or completed asynchronous call to the`OTConnect` function. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTRcvConnect (
   EndpointRef ref,
   TCall *call
);
```

Parameters

ref

call

Return Value

A result code. See "Open Transport Result Codes" (page 2722).

Discussion

You call the OTRcvConnect function to determine the status of a previously issued OTConnect call. If you want to retrieve information about the connection, you must allocate buffers for the addr field and, if required, the opt and udata fields before you make the call.

If the endpoint is synchronous and blocking, the OTRcvConnect function waits for the connection to be accepted or rejected. If the connection is accepted, the function returns with a kOTNoError result. If the connection is rejected, the function returns with a kOTLookErr result. In this case, you should call the OTLook function to verify that a T_DISCONNECT event is the reason for the kOTLookErr, and then you should call the OTRcvDisconnect function to clear the event.

If the endpoint is asynchronous or nonblocking, the OTRcvConnect function returns with the kOTNoDataErr result if the connection has not yet been established.

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTRcvDisconnect

Identifies the cause of a connection break or of a connection rejection, acknowledges and clears the corresponding disconnection event. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTRcvDisconnect (
   EndpointRef ref,
   TDiscon *discon
);
```

Parameters

ref

discon

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
Calling the OTRcvDisconnect function clears the corresponding disconnection event and retrieves any user data sent with the disconnection.

If you do not care about data returned with the disconnection and do not needto know the reason for the disconnection nor the sequence ID, you may specify a nil pointer for the discon parameter. In this case, the provider discards any user data associated with the disconnection.

The OTRcvDisconnect function behaves in the same way for all modes of operation. If there is no disconnection request pending, the function returns with the kOTNoDisconnectErr result. If there is a disconnection request pending, the function returns either the kOTNoError or kOTBufferOverflowErr result. In the latter instance, you need to check the discon field of the `TEndpointInfo` (page 2542) structure for your endpoint and make sure that the buffer referenced by the udata.buf field is at least as big as the value specified for the discon field.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTRcvOrderlyDisconnect

Acknowledges a request for an orderly disconnect. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTRcvOrderlyDisconnect (
   EndpointRef ref
);
```

Parameters

ref

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
The OTRcvOrderlyDisconnect function is a service that is not supported by all protocols. If it is, the servtype field of the `TEndpointInfo` (page 2542) structure has the value `T_COTS_ORD` or `T_TRANS_ORD` for the endpoint.

After using the OTRcvOrderlyDisconnect function to acknowledge receipt of a disconnection request, there will not be any more data to receive. Calls to the OTRcv function (for a transactionless connection) or to the OTRcvRequest function (for a transaction-based connection) after acknowledging a disconnection request fail with the kOTOutStateErr result. If the endpoint supports a remote orderly disconnect, you can still send data over the connection if you have not yet called the OTSndOrderlyDisconnect function.

The OTRcvOrderlyDisconnect function behaves in the same way in all modes of operation. If there is no disconnection request pending, the function returns with the kOTNoReleaseErr result. It there is a disconnection request pending, the function returns either the kOTNoError or kOTBufferOverflowErr result. In the latter instance, you need to check the discon field of the`TEndpointInfo` (page 2542) structure for your endpoint and make sure that the buffer referenced by the udata.buf field is at least as big as the value specified for the discon field.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTRcvUData

Reads data sent by a client using a connectionless transactionless protocol. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTRcvUData (
   EndpointRef ref,
   TUnitData *udata,
   OTFlags *flags
);
```

Parameters
ref
udata
flags

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
When the OTRcvUData function returns, it passes a pointer to a TUnitData structure containing information about the data read and a pointer to a flags variable that is set to indicate whether there is more data to be retrieved. If the buffer pointed to by the udata->udata.buf field is not large enough to hold the current data unit, the endpoint provider fills the buffer and sets the flags parameter to T_MORE to indicate that you must call the OTRcvUData function again to receive additional data. Subsequent calls to the OTRcvUData function return 0 for the length of the address and option buffers until you receive the full data unit. The last unit to be received does not have the T_MORE flag set.

If the endpoint is in asynchronous mode or is not blocking and data is not available, the OTRcvUData function fails with the kOTNoDataErr result. The endpoint provider uses the T_DATA event to notify the endpoint when data becomes available. You can use a notifier function or the OTLook function to retrieve the event. Once you get the T_DATA event, you should continue calling the OTRcvUData function until it returns the kOTNoDataErr result.

It is possible that the provider generates an erroneous T_DATA event. This is the case when the provider calls your notifier, passing T_DATA for the code parameter; but when you execute the OTRcvUData function, it returns with a kOTNoDataErr result. If this happens, you should continue normal processing and assume that the next T_DATA event is genuine.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTRcvUDErr

Clears an error condition indicated by a `T_UDERR` event and returns the reason for the error. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTRcvUDErr (
   EndpointRef ref,
   TUDErr *uderr
);
```

Parameters
ref
uderr

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
You use the OTRcvUDErr function if you have called the OTSndUData function and the endpoint provider has issued the T_UDERR event to indicate that the send operation did not succeed. This usually happens when the endpoint provider cannot determine immediately that you have specified a bad address or option value. For example, assume that you are using AppleTalk and you specify an NBP address. If Open Transport cannot resolve the address, it sends a T_UDERR event to your notifier function. To clear the error condition and determine the cause of the failure, you must call the OTRcvUDErr function.

If the size of the option or error data returned exceeds the size of the allocated buffers, the OTRcvUDErr function returns with the result kOTBufferOverflowErr, but the error indication is cleared anyway.

If you do not need to identify the cause of the failure, you can set the uderr pointer to nil. In this case, the OTRcvUDErr function clears the error indication without reporting any information to you. It is important, nevertheless, that you actually call the OTRcvUDErr function to clear the error condition. If you don't call this function, the endpoint remains in an invalid state for doing other send operations, and the endpoint provider is unable to deallocate memory reserved for internal buffers associated with the send operation.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
OpenTransport.h

OTReadBuffer

Copies data out of a no-copy receive buffer. (Deprecated in Mac OS X v10.4.)

```
Boolean OTReadBuffer (
   OTBufferInfo *buffer,
   void *dest,
   OTByteCount *len
);
```

Parameters

buffer

dest

len

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
OpenTransportProtocol.h

OTRegisterAsClientInContext

Registers your application as a client of Open Transport and gives Open Transport a notifier function it can use to send you events. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTRegisterAsClientInContext (
   OTClientName name,
   OTNotifyUPP proc,
   OTClientContextPtr clientContext
);
```

Parameters

name

proc

clientContext

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransport.h`

OTRegisterName

Registers an entity name on the network. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTRegisterName (
   MapperRef ref,
   TRegisterRequest *req,
   TRegisterReply *reply
);
```

Parameters

ref

req

reply

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
If the name-registration protocol defined using the config parameter to the OTOpenMapper or OTAsyncOpenMapper function supports dynamic name and address registration, you can use the OTRegisterName function to make a name visible on the network to other network devices.

Some protocol implementations under Open Transport allow a client to specify a name rather than an address when binding the endpoint using the OTBind function. Binding an endpoint by name causes the protocol to automatically register the name on the network if the protocol supports dynamic name registration. This is the simpler technique for registering a name and is preferred over creating a mapper provider and then using the OTRegisterName function to register the name.

The format for the requested name and address is specific to the protocol used. Please consult the documentation for the protocol you are using for format information.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTReleaseBuffer

Returns the no-copy receive buffer to the system. (Deprecated in Mac OS X v10.4.)

```
void OTReleaseBuffer (
   OTBuffer *buffer
);
```

Parameters

buffer

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
OpenTransportProtocol.h

OTRemoveFirst

Removes the first link in a FIFO list. (Deprecated in Mac OS X v10.4.)

```
OTLink * OTRemoveFirst (
   OTList *list
);
```

Parameters

list

Return Value
See the description of the `OTLink` data type.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
OpenTransport.h

OTRemoveLast

Removes the last link in a FIFO list. (Deprecated in Mac OS X v10.4.)

```
OTLink * OTRemoveLast (
   OTList *list
);
```

Parameters

list

Return Value
See the description of the `OTLink` data type.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTRemoveLink

Removes the last link in a FIFO list. (Deprecated in Mac OS X v10.4.)

```
Boolean OTRemoveLink (
   OTList *list,
   OTLink *link
);
```

Parameters
list
link

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTRemoveNotifier

Removes a provider's notifier function. (Deprecated in Mac OS X v10.4.)

```
void OTRemoveNotifier (
   ProviderRef ref
);
```

Parameters
ref

Discussion
The `OTRemoveNotifier` function removes the notifier (if any) currently installed for the provider specified by the `ref` parameter.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
OpenTransport.h

OTResolveAddress

Returns the protocol address that corresponds to the name of an endpoint. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTResolveAddress (
   EndpointRef ref,
   TBind *reqAddr,
   TBind *retAddr,
   OTTimeout timeOut
);
```

Parameters

ref

reqAddr

retAddr

timeOut

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
The OTResolveAddress function returns the lowest-level address for your endpoint. Not all endpoints support this function. A value of CAN_RESOLVE_ADDR in the flags field of the TEndpointInfo (page 2542) structure indicates that the endpoint does support this function. Using this function saves you the trouble of opening and closing a mapper provider if the only reason you have for opening the mapper is to look up the address corresponding to a specific endpoint name. You would still have to open the mapper if you needed to look up a name pattern—that is, if the name included any wildcard characters.

You are responsible for initializing the buffers described by the req and ret parameters required to hold the addresses. To determine how large these buffers should be, examine the addr field of the TEndpointInfo structure, which specifies the maximum amount of memory needed to store an address for the endpoint specified by the ref parameter.

If a notifier is not installed, it is not possible to determine when the OTResolveAddress function completes.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
OpenTransport.h

OTReverseList

Reverses the order in which entries are linked in a list. (Deprecated in Mac OS X v10.4.)

```
OTLink * OTReverseList (
   OTLink *list
);
```

Parameters

list

Return Value
See the description of the `OTLink` data type.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTScheduleDeferredTask

Schedules a task for execution at deferred task time. (Deprecated in Mac OS X v10.4.)

```
Boolean OTScheduleDeferredTask (
   OTDeferredTaskRef dtCookie
);
```

Parameters

dtCookie

Discussion
The `OTScheduleDeferredTask` function schedules for execution at the next deferred task time the task associated with the `dtCookie` parameter, which is the reference returned by the `OTCreateDeferredTask` function.

You can call this function at any time. If you have not yet destroyed a task, you can use this function to reschedule the same task more than once.

If you makes multiple calls to the `OTScheduleDeferredTask` function before the task is executed, additional tasks are not scheduled; only one instance of each unique task can only be scheduled at a time.

This function returns `true` if it scheduled the deferred task successfully, `false` if not. If it returns `false` and the `dtCookie` parameter has a valid value (other than 0), then the task is already scheduled to run. If `dtCookie` is invalid (a value of 0), the function returns `false` and does nothing.

If you want to call Open Transport from an interrupt, you can use this function (and the `OTCreateDeferredTask` function) instead of the standard Deferred Task Manager function `DTInstall` to create a deferred task that permits you to call Open Transport function calls. This allows Open Transport to adapt to changes in the underlying operating system without affecting the client's code.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTScheduleTimerTask

Schedules a timer task to be executed at the specified time. (Deprecated in Mac OS X v10.4.)

```
Boolean OTScheduleTimerTask (
   OTTimerTask timerTask,
   OTTimeout milliSeconds
);
```

Parameters

timerTask

milliSeconds

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProtocol.h`

OTSetAddressFromNBPEntity

Stores an NBP entity structure as an NBP address string. (Deprecated in Mac OS X v10.4.)

```
OTByteCount OTSetAddressFromNBPEntity (
   UInt8 *nameBuf,
   const NBPEntity *entity
);
```

Parameters

nameBuf

A pointer to the NBP address buffer in which you wish to store the NBP entity.

entity

Return Value
See the description of the `OTByteCount` data type.

Discussion
The `OTSetAddressFromNBPEntity` function stores the information in the NBP entity into the buffer specified by the `nameBuf` parameter in the format required for mapper calls—that is, if you have a backslash (\), a colon (:), or an at-sign (@) in your NBP name, this function inserts a backslash before each so that the mapper functions can handle them correctly. This function returns the number of bytes that were actually used in the buffer.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTSetAddressFromNBPString

Copies an NBP name string into an NBP address buffer. (Deprecated in Mac OS X v10.4.)

```
OTByteCount OTSetAddressFromNBPString (
   UInt8 *addrBuf,
   const char *name,
   SInt32 len
);
```

Parameters

addrBuf
A pointer to the NBP address buffer in which to store the NBP name string.

name
A pointer to the NBP name string you wish to copy into the buffer.

len
The number of characters to copy.

Return Value
See the description of the `OTByteCount` data type.

Discussion
The `OTSetAddressFromNBPString` function copies the string indicated by the `nbpName` parameter into the buffer indicated by the `addrBuf` parameter. The `len` parameter indicates the number of characters to copy. A value of -1 copies the entire `nbpName` string. The function returns the number of bytes actually copied.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTSetAsynchronous

Sets a provider's mode of execution to asynchronous. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTSetAsynchronous (
   ProviderRef ref
);
```

Parameters

ref

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
The `OTSetAsynchronous` function causes all functions for the provider specified in the `ref` parameter to run asynchronously. You must install a notifier function for the provider if it needs to receive completion events. You can install a notifier function either before or after calling the `OTSetAsynchronous` function.

Changing a provider's mode of execution does not affect its notifier function, if any; the notifier function remains installed.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTSetBit

Sets a bit atomically. (Deprecated in Mac OS X v10.4.)

```
Boolean OTSetBit (
   UInt8 *bitMap,
   OTByteCount bitNo
);
```

Parameters

bitMap

bitNo

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransportProtocol.h`

OTSetBlocking

Allows a provider to wait or block until it is able to send or receive data. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTSetBlocking (
   ProviderRef ref
);
```

Parameters

ref

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
The `OTSetBlocking` function causes provider functions that send or receive data to wait if current conditions prevent them from completing an operation. By default, a provider is in nonblocking mode, in which case, if a provider function were unable to complete sending or receiving data, it would return immediately with a result that would tell you why the operation was unable to complete.

If a provider is in blocking mode and you call the `OTCloseProvider` function to close the provider, Open Transport gives each Streams module up to 15 seconds to process outgoing commands. It is recommended that you call the `OTSetNonBlocking` function before you call the `OTCloseProvider` function.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTSetBusTypeInPortRef

Sets bus type for a port reference. (Deprecated in Mac OS X v10.4.)

```
OTPortRef OTSetBusTypeInPortRef (
   OTPortRef ref,
   OTBusType busType
);
```

Parameters

ref

busType

Return Value
See the description of the `OTPortRef` data type.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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.

Carbon Porting Notes

OT ports are read only in Carbon. In Mac OS X, code that communicates directly with network interfaces must use the IOKit API.

Declared In
`OpenTransport.h`

OTSetDeviceTypeInPortRef

Sets device type for a port reference. (Deprecated in Mac OS X v10.4.)

```
OTPortRef OTSetDeviceTypeInPortRef (
   OTPortRef ref,
   OTDeviceType devType
);
```

Parameters

ref

devType

Return Value
See the description of the `OTPortRef` data type.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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.

Carbon Porting Notes

OT ports are read only in Carbon. In Mac OS X, code that communicates directly with network interfaces must use the IOKit API.

Declared In
`OpenTransport.h`

OTSetFirstClearBit

Atomcially sets the first clear bit in a specified bit map. (Deprecated in Mac OS X v10.4.)

```
OTResult OTSetFirstClearBit (
   UInt8 *bitMap,
   OTByteCount startBit,
   OTByteCount numBits
);
```

Parameters

bitMap

startBit

numBits

Return Value
See the description of the `OTResult` data type.

Discussion
Sets the first clear bit in `bitMap` , starting with `startBit` and giving up after `numBits`. Returns the bit number that was set, or a `kOTNotFoundErr` error if there was no clear bit available

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransportProtocol.h`

OTSetNBPEntityFromAddress

Parses and stores an NBP address into an NBP entity. (Deprecated in Mac OS X v10.4.)

```
Boolean OTSetNBPEntityFromAddress (
   NBPEntity *entity,
   const UInt8 *addrBuf,
   OTByteCount len
);
```

Parameters

entity

addrBuf

A pointer to the address buffer in which to store the NBP name string.

len

Discussion
The `OTSetNBPEntityFromAddress` function parses an NBP address or a combined DDP-NBP address into the NBP name's constituent parts (name, type, and zone) and stores the result in an NBP entity. The function ignores the DDP address part of a combined DDP-NBP address. From the NBP entity, each of the constituent parts of the name can be later retrieved or changed.

This function returns `true` if it worked successfully; it returns `false` if it had to truncate any data—that is, if the address had data that was too long in one of the fields, each of which only holds 32 characters of data. When this occurs, Open Transport still stores the data, but in a truncated form.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransportProviders.h`

OTSetNBPName

Stores the name part of an NBP name into an NBP entity structure. (Deprecated in Mac OS X v10.4.)

```
Boolean OTSetNBPName (
   NBPEntity *entity,
   const char *name
);
```

Parameters

entity

name

A pointer to the name portion of an NBP name string that you wish to store.

Discussion

The `OTSetNBPName` function stores the NBP name specified by the `name` parameter into the NBP entity structure indicated by the `nbpEntity` parameter, deleting any previous name stored there. This function returns `false` if the `name` parameter is longer than the maximum allowed for a name part of an NBP name (32 characters).

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`OpenTransportProviders.h`

OTSetNBPType

Stores the type part of an NBP name in an NBP entity structure. (Deprecated in Mac OS X v10.4.)

```
Boolean OTSetNBPType (
   NBPEntity *entity,
   const char *typeVal
);
```

Parameters

entity

typeVal

A pointer to the type portion of an NBP name string that you wish to store.

Discussion

The `OTSetNBPType` function stores the NBP type specified by the `type` parameter into the NBP entity structure indicated by the `nbpEntity` parameter, deleting any previous type stored there. The type supplied must not have any escape characters stored in it, although you do not receive any error message if you do use such characters. This function returns `false` if the `type` parameter is longer than the maximum allowed for type part of an NBP name (32 characters).

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`OpenTransportProviders.h`

OTSetNBPZone

Stores the zone part of an NBP name in an NBP entity structure. (Deprecated in Mac OS X v10.4.)

```
Boolean OTSetNBPZone (
   NBPEntity *entity,
   const char *zone
);
```

Parameters

entity

zone

A pointer to the zone portion of an NBP name string that you wish to store.

Discussion

The `OTSetNBPZone` function stores the NBP zone specified by the `zone` parameter into the NBP entity structure indicated by the `nbpEntity` parameter, deleting any previous zone stored there. The zone supplied must not have any of the NBP escape characters stored in it, although you do not receive any error message if you do use such characters. This function returns `false` if the zone parameter is longer than the maximum allowed for zone part of an NBP name (32 characters).

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`OpenTransportProviders.h`

OTSetNonBlocking

Disallows a provider from waiting if it cannot currently complete a function that sends or receives data. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTSetNonBlocking (
   ProviderRef ref
);
```

Parameters

ref

Return Value

A result code. See "Open Transport Result Codes" (page 2722).

Discussion

The `OTSetNonBlocking` function causes provider functions to return a result code immediately, instead of waiting for a function that sends or receives data to complete. When you open a provider, its mode of operation is set to nonblocking by default.

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTSetSynchronous

Sets a provider's mode of execution to synchronous. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTSetSynchronous (
   ProviderRef ref
);
```

Parameters

ref

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
The `OTSetSynchronous` function causes all provider functions to run synchronously when using the provider that you specify.

Changing a provider's mode of execution does not affect its notifier function, if any is installed for this provider; the notifier function remains installed.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTSnd

Sends data to a remote peer. (Deprecated in Mac OS X v10.4.)

```
OTResult OTSnd (
   EndpointRef ref,
   void *buf,
   OTByteCount nbytes,
   OTFlags flags
);
```

Parameters

ref

buf

A pointer to the data being sent. If you are sending data that is not stored contiguously, this is a pointer to an OTData structure that describes the first data fragment.

nbytes
flags

Return Value
See the description of the `OTResult` data type.

Discussion
You use the OTSnd function to send data to a remote peer. Before you use this function, you must establish a connection with the peer.

If the OTSnd function succeeds, it returns an integer (OSStatus) specifying the number of bytes that were actually sent. If it fails, it returns a negative integer corresponding to a result code that indicates the reason for the failure.

You specify the data to be sent by passing a pointer to the data (buf) and byspecifying the size of the data (nbytes). The maximum size of the data you can send is specified by the `tsdu` field of the `TEndpointInfo` (page 2542) structure for the endpoint.

Some protocols use expedited data for control or attention messages. To determine whether the endpoint supports this service, examine the `etsdu` field of the `TEndpointInfo` structure. A positive integer for the `etsdu` field indicates the maximum size in bytes of expedited data that you can send. To send expedited data, you must set the T_EXPEDITED bit of the flags parameter.

If you want to break up the data sent into smaller logical units, you can set the T_MORE bit of the flags parameter to indicate that you are using additional calls to the OTSnd function to send more data that belongs to the same logical unit. To indicate that the last data unit is being sent, you must specify 0 for nbytes and turn off the T_MORE flag. This is the only circumstance under which it is permitted to send a zero-length data unit. If the endpoint does not support the sending of zero-length data, the OTSnd function fails with the kOTBadDataErr result.

If the endpoint is in blocking mode, the OTSnd function returns after it actually sends the data. If flow-control restrictions prevent its sending the data, it retries the operation until it is able to send it. If the endpoint is in nonblocking mode, the OTSnd function returns with the kOTFlowErr result if flow-control restrictions prevent the data from being sent. When the endpoint provider is able to send the data, it returns a T_GODATA event to let you know that it is possible to send data.

The following table shows how the endpoint's mode of execution and blocking status affects the behavior of the OTSnd function.

Table 44-1

	Blocking	Nonblocking
Synchronous	The function returns when the provider lifts flow-control restrictions The kOTFlowErr result is never returned.	The function returns immediately. The kOTFlowErr result might be returned.
Asynchronous	The function returns immediately. The kOTFlowErr result is never returned.	The function returns immediately. The kOTFlowErr result might be returned.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
OpenTransport.h

OTSndDisconnect

Tears down an open connection (abortive disconnect) or rejects an incoming connection request. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTSndDisconnect (
   EndpointRef ref,
   TCall *call
);
```

Parameters

ref

call

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
There are two functions that you can use to tear down a connection: OTSndDisconnect for an abortive disconnect, or OTSndOrderlyDisconnect for an orderly disconnect. It is recommended that you use the OTSndOrderlyDisconnect function for tearing down a connection whenever possible and that you use the OTSndDisconnect function only for rejecting incoming connection requests.

If the endpoint is in asynchronous mode, the OTSndDisconnect function returns immediately with a result of kOTNoError to indicate that the disconnection process has begun and that your notifier function will be called when the process completes.

When the connection has been broken, the provider issues a T_DISCONNECTCOMPLETE event. If you have installed a notifier function, Open Transport calls your notifier and passes this event in the code parameter. The cookie parameter contains the call parameter. If you have not installed a notifier function, you cannot determine when this function completes.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
OpenTransport.h

OTSndOrderlyDisconnect

Initiates or completes an orderly disconnection. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTSndOrderlyDisconnect (
   EndpointRef ref
);
```

Parameters

ref

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
You call the `OTSndOrderlyDisconnect` function to initiate an orderly release of a connection and to indicate to the peer endpoint that you have no more data to send. After calling this function, you must not send any more data over the connection. However, you can still continue to receive data if the peer endpoint has not yet called the `OTSndOrderlyDisconnect` function.

This function is a service that is not supported by all protocols. If it is supported, the servtype field of the `TEndpointInfo` (page 2542) structure has the value `T_COTS_ORD` or `T_TRANS_ORD`.

The `OTSndOrderlyDisconnect` function behaves exactly the same in all modes of operation.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTSndUData

Sends data using a connectionless transactionless endpoint. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTSndUData (
   EndpointRef ref,
   TUnitData *udata
);
```

Parameters

ref

udata

A pointer to a TUnitData structure that specifies thedata to be sent and its destination.

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
If the endpoint is in synchronous, blocking mode, the OTSndUData function returns immediately. If flow-control restrictions prevent its sending the data, it retries the operation until it is able to send it. If the endpoint is in nonblocking mode, the OTSndUData function returns a kOTFlowErr result if flow-control restrictions prevent the data from being sent. When the endpoint provider is able to send the data, it calls your notifier function, passing T_GODATA for the code parameter. You can then call the OTSndUData function from your notifier to send the data. You can also retrieve this event by polling the endpoint using the OTLook function.

Some endpoint providers are not able to detect immediately whether you specified incorrect address or option information. In such cases, the provider calls your notifier function when it detects the error, passing the T_UDERR for the code parameter to advise you that an error has occurred. You can determine the cause of this event by calling the OTRcvUDErr function and examining the value of the uderr->error parameter. It is important that you call the OTRcvUDErr function even if you are not interested in examining the cause of the error. Failing to do this leaves the endpoint in an invalid state for doing other sends and makes the endpoint provider unable to deallocate memory reserved for internal buffers associated with the send.

The next table shows how the endpoint's mode of execution and blocking status affects the behavior of the OTSndUData function.

Table 44-2

	Blocking	Nonblocking
Synchronous	The function returns when the provider lifts flow-control restrictions The kOTFlowErr result is never returned.	The function returns immediately. the kOTFlowErr result might be returned
Asynchronous	The function returns immediately. The kOTFlowErr result is never returned.	the function returns immediately. the kOTFlowErr result might be returned.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTStrCat

Concatenates two C strings. (Deprecated in Mac OS X v10.4.)

```
void OTStrCat (
   char *dest,
   const char *src
);
```

Parameters
dest
src

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTStrCopy

Copies a C string. (Deprecated in Mac OS X v10.4.)

```
void OTStrCopy (
   char *dest,
   const char *src
);
```

Parameters

dest

src

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTStrEqual

Determines whether two C strings are the same. (Deprecated in Mac OS X v10.4.)

```
Boolean OTStrEqual (
   const char *src1,
   const char *src2
);
```

Parameters

src1

src2

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTStrLength

Returns the length of a C string. (Deprecated in Mac OS X v10.4.)

```
OTByteCount OTStrLength (
   const char *str
);
```

Parameters

str

Return Value
See the description of the `OTByteCount` data type.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTSubtractTimeStamps

Subtracts one timestamp value from another. (Deprecated in Mac OS X v10.4.)

```
OTTimeStamp * OTSubtractTimeStamps (
   OTTimeStamp *result,
   OTTimeStamp *startTime,
   OTTimeStamp *endEnd
);
```

Parameters

result
startTime
endEnd

Return Value
See the description of the `OTTimeStamp` data type.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTTestBit

Atomically tests a bit in a specified bit map. (Deprecated in Mac OS X v10.4.)

```
Boolean OTTestBit (
   UInt8 *bitMap,
   OTByteCount bitNo
);
```

Parameters

bitMap

bitNo

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransportProtocol.h`

OTTimeStampInMicroseconds

Calculates the time elapsed in microseconds since since a specified time. (Deprecated in Mac OS X v10.4.)

```
UInt32 OTTimeStampInMicroseconds (
   OTTimeStamp *delta
);
```

Parameters

delta

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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

`OpenTransport.h`

OTTimeStampInMilliseconds

Calculates the time elapsed in milliseconds since since a specified time. (Deprecated in Mac OS X v10.4.)

```
UInt32 OTTimeStampInMilliseconds (
   OTTimeStamp *delta
);
```

Parameters

delta

Availability

Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTUnbind

Dissociates an endpoint from its address or cancels an asynchronous call to the `OTBind` function. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTUnbind (
   EndpointRef ref
);
```

Parameters

ref

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Discussion
If you call the `OTUnbind` function asynchronously and you have not installed a notifier function, the only way to determine that the endpoint has been unbound is to use the OTGetEndpointState function to poll the state of the endpoint. The function returns the kOTStateChangeErr result when the OTUnbind function returns. If the function succeeds, the state of the endpoint is T_UNBND. If it fails, its state is T_IDLE.

After you unbind an endpoint, you can no longer use it to send or receive information. You can use the OTCloseProvider function to deallocate memory reserved for the endpoint, or you can use the OTBind function to associate it with another address and then resume transferring data or establishing a connection.

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

OTUnregisterAsClientInContext

Removes your application as a client of Open Transport. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTUnregisterAsClientInContext (
   OTClientContextPtr clientContext
);
```

Parameters

clientContext

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`OpenTransport.h`

OTUseSyncIdleEvents

Allows synchronous idle events to be sent to your notifier. (Deprecated in Mac OS X v10.4.)

```
OSStatus OTUseSyncIdleEvents (
   ProviderRef ref,
   Boolean useEvents
);
```

Parameters

ref

useEvents

Return Value
A result code. See "Open Transport Result Codes" (page 2722).

Availability
Available in CarbonLib 1.0 and later when OpenTransport 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
`OpenTransport.h`

Callbacks by Task

Notifier Callbacks

`OTNotifyProcPtr` (page 2419)

System, Timer, and Deferred Task Callbacks

`OTProcessProcPtr` (page 2420)

Linked List Callbacks

Miscellaneous Callbacks

Callbacks

admin_t

```
typedef OTInt32 (*admin_t) ();
```

If you name your function `MyAdmin_tCallback`, you would declare it like this:

```
OTInt32 MyAdmin_tCallback ();
```

Parameters

Return Value

See the description of the `OTInt32` data type.

Carbon Porting Notes

Carbon does not support any STREAMS functionality because the STREAMS subsystem is not available on Mac OS X.

bufcallp_t

```
typedef void (*bufcallp_t) (
    SInt32 size
);
```

If you name your function `MyBufcallp_tCallback`, you would declare it like this:

```
void MyBufcallp_tCallback (
    SInt32 size
);
```

Parameters

size

Carbon Porting Notes

Carbon does not support any STREAMS functionality because the STREAMS subsystem is not available on Mac OS X.

bufcall_t

```
typedef void (*bufcall_t) (
    SInt32 size
);
```

If you name your function `MyBufcall_tCallback`, you would declare it like this:

```
void MyBufcall_tCallback (
    SInt32 size
);
```

Parameters

size

Carbon Porting Notes

Carbon does not support any STREAMS functionality because the STREAMS subsystem is not available on Mac OS X.

closeOld_t

```
typedef OTInt32 (*closeOld_t) (
    queue *q
);
```

If you name your function `MyCloseOld_tCallback`, you would declare it like this:

```
OTInt32 MyCloseOld_tCallback (
    queue *q
);
```

Parameters

q

Return Value

See the description of the `OTInt32` data type.

Carbon Porting Notes

Carbon does not support any STREAMS functionality because the STREAMS subsystem is not available on Mac OS X.

closep_t

```
typedef OTInt32 (*closep_t) (
    queue *q,
    OTInt32 foo,
    cred_t *cred
);
```

If you name your function `MyClosep_tCallback`, you would declare it like this:

```
OTInt32 MyClosep_tCallback (
    queue *q,
    OTInt32 foo,
    cred_t *cred
);
```

Parameters

q

foo

cred

Return Value

See the description of the `OTInt32` data type.

Carbon Porting Notes

Carbon does not support Open Transport configuration APIs because the Mac OS X networking stack is not based on STREAMS.

esbbcallProc

```
typedef void (*esbbcallProc) (
    SInt32 arg
);
```

If you name your function `MyEsbbcallCallback`, you would declare it like this:

```
void MyEsbbcallCallback (
    SInt32 arg
);
```

Parameters

arg

Carbon Porting Notes

This function is not needed in Carbon because the STREAMS subsystem is not available on Mac OS X.

FreeFuncType

```
typedef void (*FreeFuncType) (
    char *arg
);
```

If you name your function `MyFreeFuncTypeCallback`, you would declare it like this:

```
void MyFreeFuncTypeCallback (
    char *arg
);
```

Parameters

arg

Carbon Porting Notes

This function is not needed in Carbon because the STREAMS subsystem is not available on Mac OS X.

old_closep_t

```
typedef OTInt32 (*old_closep_t) (
    queue *q
);
```

If you name your function `MyOld_closep_tCallback`, you would declare it like this:

```
OTInt32 MyOld_closep_tCallback (
    queue *q
);
```

Parameters

q

Return Value

See the description of the `OTInt32` data type.

Carbon Porting Notes

Carbon does not support any STREAMS functionality because the STREAMS subsystem is not available on Mac OS X.

old_openp_t

```
typedef OTInt32 (*old_openp_t) (
    queue *q,
    dev_t dev,
    OTInt32 foo,
    OTInt32 bar
);
```

If you name your function `MyOld_openp_tCallback`, you would declare it like this:

```
OTInt32 MyOld_openp_tCallback (
```

```
    queue *q,
    dev_t dev,
    OTInt32 foo,
    OTInt32 bar
);
```

Parameters

q

dev

foo

bar

Return Value
See the description of the `OTInt32` data type.

Carbon Porting Notes

Carbon does not support any STREAMS functionality because the STREAMS subsystem is not available on Mac OS X.

openOld_t

```
typedef OTInt32 (*openOld_t) (
    queue *q,
    dev_t dev,
    OTInt32 foo,
    OTInt32 bar
);
```

If you name your function `MyOpenOld_tCallback`, you would declare it like this:

```
OTInt32 MyOpenOld_tCallback (
    queue *q,
    dev_t dev,
    OTInt32 foo,
    OTInt32 bar
);
```

Parameters

q

dev

foo

bar

Return Value
See the description of the `OTInt32` data type.

Carbon Porting Notes

Carbon does not support any STREAMS functionality because the STREAMS subsystem is not available on Mac OS X.

openp_t

```
typedef OTInt32 (*openp_t) (
    queue *q,
    dev_t *dev,
    OTInt32 foo,
    OTInt32 bar,
    cred_t *cred
);
```

If you name your function `MyOpenp_tCallback`, you would declare it like this:

```
OTInt32 MyOpenp_tCallback (
    queue *q,
    dev_t *dev,
    OTInt32 foo,
    OTInt32 bar,
    cred_t *cred
);
```

Parameters

q
dev
foo
bar
cred

Return Value

See the description of the `OTInt32` data type.

Carbon Porting Notes

Carbon does not support any STREAMS functionality because the STREAMS subsystem is not available on Mac OS X.

OTAllocMemProcPtr

```
typedef void (*OTAllocMemProcPtr) (
    OTByteCount size
);
```

If you name your function `MyOTAllocMemProc`, you would declare it like this:

```
void MyOTAllocMemProc (
    OTByteCount size
);
```

Parameters

size

Carbon Porting Notes

Carbon does not support any STREAMS functionality because the STREAMS subsystem is not available on Mac OS X.

OTCanConfigureProcPtr

```
typedef Boolean (*OTCanConfigureProcPtr)
(
    OTConfigurationRef cfig,
    UInt32 pass
);
```

If you name your function `MyOTCanConfigureProc`, you would declare it like this:

```
Boolean MyOTCanConfigureProc (
    OTConfigurationRef cfig,
    UInt32 pass
);
```

Parameters

cfig

pass

Carbon Porting Notes

Carbon does not support access to Open Transport configuration APIs because the Mac OS X networking stack is not based on STREAMS.

OTCFConfigureProcPtr

```
typedef OSStatus (*OTCFConfigureProcPtr)
(
    TOTConfiguratorRef cfigor,
    OTConfigurationRef cfig
);
```

If you name your function `MyOTCFConfigureProc`, you would declare it like this:

```
OSStatus MyOTCFConfigureProc (
    TOTConfiguratorRef cfigor,
    OTConfigurationRef cfig
);
```

Parameters

cfigor

cfig

Return Value

A result code. See "Open Transport Result Codes" (page 2722).

Carbon Porting Notes

Carbon does not support access to Open Transport configuration APIs because the Mac OS X networking stack is not based on STREAMS.

OTCFCreateStreamProcPtr

```
typedef OSStatus (*OTCFCreateStreamProcPtr)
(
    TOTConfiguratorRef cfigor,
    OTConfigurationRef cfig,
    OTOpenFlags oFlags,
    OTNotifyUPP proc,
    void *contextPtr
);
```

If you name your function `MyOTCFCreateStreamProc`, you would declare it like this:

```
OSStatus MyOTCFCreateStreamProc (
    TOTConfiguratorRef cfigor,
    OTConfigurationRef cfig,
    OTOpenFlags oFlags,
    OTNotifyUPP proc,
    void *contextPtr
);
```

Parameters

cfigor

cfig

oFlags

proc

contextPtr

Return Value

A result code. See "Open Transport Result Codes" (page 2722).

Carbon Porting Notes

Carbon does not support access to Open Transport configuration APIs because the Mac OS X networking stack is not based on STREAMS.

OTCFHandleSystemEventProcPtr

```
typedef void (*OTCFHandleSystemEventProcPtr)
(
    TOTConfiguratorRef cfigor,
    OTEventCode code,
    OTResult result,
    void *cookie
);
```

If you name your function `MyOTCFHandleSystemEventProc`, you would declare it like this:

```
void MyOTCFHandleSystemEventProc (
    TOTConfiguratorRef cfigor,
    OTEventCode code,
    OTResult result,
    void *cookie
);
```

Parameters

cfigor

code

result

cookie

Carbon Porting Notes

Carbon does not support access to Open Transport configuration APIs because the Mac OS X networking stack is not based on STREAMS.

OTCreateConfiguratorProcPtr

```
typedef OSStatus (*OTCreateConfiguratorProcPtr)
(
    TOTConfiguratorRef *cfigor
);
```

If you name your function `MyOTCreateConfiguratorProc`, you would declare it like this:

```
OSStatus MyOTCreateConfiguratorProc (
    TOTConfiguratorRef *cfigor
);
```

Parameters

cfigor

Return Value

A result code. See "Open Transport Result Codes" (page 2722).

Carbon Porting Notes

Carbon does not support access to Open Transport configuration APIs because the Mac OS X networking stack is not based on STREAMS.

OTGateProcPtr

```
typedef Boolean (*OTGateProcPtr) (
    OTLink *thisLink
);
```

If you name your function `MyOTGateProc`, you would declare it like this:

```
Boolean MyOTGateProc (
    OTLink *thisLink
);
```

Parameters

thisLink

Carbon Porting Notes

Carbon does not support Open Transport configuration APIs because the Mac OS X networking stack is not based on STREAMS.

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

Declared In
`OpenTransportProtocol.h`

OTGetPortIconProcPtr

```
typedef Boolean (*OTGetPortIconProcPtr)
(
    OTPortRecord *port,
    OTResourceLocator *iconLocation
);
```

If you name your function `MyOTGetPortIconProc`, you would declare it like this:

```
Boolean MyOTGetPortIconProc (
    OTPortRecord *port,
    OTResourceLocator *iconLocation
);
```

Parameters

port

iconLocation

Carbon Porting Notes

Carbon does not support access to the Open Transport port name or icon because this information is not available on Mac OS X.

OTGetPortNameProcPtr

```
typedef void (*OTGetPortNameProcPtr)
(
    OTPortRecord *port,
    OTBooleanParam includeSlot,
    OTBooleanParam includePort,
    Str255 userVisibleName
);
```

If you name your function `MyOTGetPortNameProc`, you would declare it like this:

```
void MyOTGetPortNameProc (
    OTPortRecord *port,
    OTBooleanParam includeSlot,
    OTBooleanParam includePort,
    Str255 userVisibleName
);
```

Parameters

port
includeSlot
includePort
userVisibleName

Carbon Porting Notes

Carbon does not support access to the Open Transport port name or icon because this information is not available on Mac OS X.

OTHashProcPtr

```
typedef UInt32 (*OTHashProcPtr) (
    OTLink *linkToHash
);
```

If you name your function `MyOTHashProc`, you would declare it like this:

```
UInt32 MyOTHashProc (
    OTLink *linkToHash
);
```

Parameters

linkToHash

Carbon Porting Notes

Carbon does not support Open Transport hash lists because Apple has not identified a developer need for them.

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

Declared In
`OpenTransportProtocol.h`

OTHashSearchProcPtr

```
typedef Boolean (*OTHashSearchProcPtr)
(
    const void *ref,
    OTLink *linkToCheck
);
```

If you name your function `MyOTHashSearchProc`, you would declare it like this:

```
Boolean MyOTHashSearchProc (
    const void *ref,
    OTLink *linkToCheck
);
```

Parameters

ref

linkToCheck

Carbon Porting Notes

Carbon does not support Open Transport hash lists because Apple has not identified a developer need for them.

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

Declared In
`OpenTransportProtocol.h`

OTListSearchProcPtr

```
typedef Boolean (*OTListSearchProcPtr)
(
    const void *ref,
    OTLink *linkToCheck
);
```

If you name your function `MyOTListSearchProc`, you would declare it like this:

```
Boolean MyOTListSearchProc (
    const void *ref,
    OTLink *linkToCheck
);
```

Parameters

ref

linkToCheck

Carbon Porting Notes

This is a function type for a user callback. Use the type `OTListSearchUPP` instead.

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

Declared In
`OpenTransport.h`

OTNotifyProcPtr

```
typedef void (*OTNotifyProcPtr) (
    void *contextPtr,
    OTEventCode code,
    OTResult result,
    void *cookie
);
```

If you name your function `MyOTNotifyProc`, you would declare it like this:

```
void MyOTNotifyProc (
    void *contextPtr,
    OTEventCode code,
    OTResult result,
    void *cookie
);
```

Parameters

contextPtr

code

result

cookie

Carbon Porting Notes

This is a function type for a callback. Use the type `OTNotifyUPP` instead.

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

Declared In
`OpenTransport.h`

OTProcessProcPtr

```
typedef void (*OTProcessProcPtr) (
    void *arg
);
```

If you name your function `MyOTProcessProc`, you would declare it like this:

```
void MyOTProcessProc (
    void *arg
);
```

Parameters

arg

Carbon Porting Notes

Use the `OTProcessUPP` type instead.

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

Declared In
`OpenTransport.h`

OTSetupConfiguratorProcPtr

```
typedef OSStatus (*OTSetupConfiguratorProcPtr)
(
    OTCanConfigureProcPtr *canConfigure,
    OTCreateConfiguratorProcPtr *createConfigurator,
    UInt8 *configuratorType
);
```

If you name your function `MyOTSetupConfiguratorProc`, you would declare it like this:

```
OSStatus MyOTSetupConfiguratorProc (
    OTCanConfigureProcPtr *canConfigure,
    OTCreateConfiguratorProcPtr *createConfigurator,
    UInt8 *configuratorType
);
```

Parameters

canConfigure

createConfigurator

configuratorType

Return Value

A result code. See "Open Transport Result Codes" (page 2722).

Carbon Porting Notes

Carbon does not support Open Transport configuration APIs because the Mac OS X networking stack is not based on STREAMS.

OTSMCompleteProcPtr

```
typedef void (*OTSMCompleteProcPtr) (
    void *contextPtr
);
```

If you name your function `MyOTSMCompleteProc`, you would declare it like this:

```
void MyOTSMCompleteProc (
    void *contextPtr
);
```

Parameters

contextPtr

Carbon Porting Notes

Carbon does not support Open Transport configuration APIs because the Mac OS X networking stack is not based on STREAMS.

OTStateProcPtr

```
typedef void (*OTStateProcPtr) (
    OTStateMachine *sm
);
```

If you name your function `MyOTStateProc`, you would declare it like this:

```
void MyOTStateProc (
    OTStateMachine *sm
);
```

Parameters

sm

Carbon Porting Notes

Carbon does not support Open Transport configuration APIs because the Mac OS X networking stack is not based on STREAMS.

putp_t

```
typedef OTInt32 (*putp_t) (
    queue *q,
    msgb *mp
);
```

If you name your function `MyPutp_tCallback`, you would declare it like this:

```
OTInt32 MyPutp_tCallback (
    queue *q,
    msgb *mp
);
```

Parameters

q

mp

Return Value

See the description of the `OTInt32` data type.

Carbon Porting Notes

Carbon does not support any STREAMS functionality because the STREAMS subsystem is not available on Mac OS X.

srvp_t

```
typedef OTInt32 (*srvp_t) (
    queue *q
);
```

If you name your function `MySrvp_tCallback`, you would declare it like this:

```
OTInt32 MySrvp_tCallback (
    queue *q
);
```

Parameters

q

Return Value
See the description of the `OTInt32` data type.

Carbon Porting Notes

Carbon does not support any STREAMS functionality because the STREAMS subsystem is not available on Mac OS X.

Data Types

AppleTalkInfo

Obtain informations about the current AppleTalk environment.

```
struct AppleTalkInfo {
    DDPAddress fOurAddress;
    DDPAddress fRouterAddress;
    UInt16 fCableRange[2];
    UInt16 fFlags;
};
typedef struct AppleTalkInfo AppleTalkInfo;
```

Fields

`fOurAddress`
The network number and node ID of your node.

`fRouterAddress`
The network number and node ID of the closest router on your network.

`fCableRange`
A two-element array indicating the first and last network numbers for the current extended network to which the machine is connected. For nonextended networks, this returns the name of the zone.

`fFlags`
A set of flag bits that describe the network. See `kATalkInfoIsExtended` (page 2612).

Discussion
Use the AppleTalk information structure to obtain information about the current AppleTalk environment for the node on which your application is running.

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

Declared In
`OpenTransportProviders.h`

ATSvcRef

```
typedef struct OpaqueATSvcRef * ATSvcRef;
```

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

Declared In
`OpenTransportProviders.h`

bandinfo

```
struct bandinfo {
    unsigned char bi_pri;
    char pad1;
    SInt32 bi_flag;
};
typedef struct bandinfo bandinfo;
```

Fields
`bi_pri`
`pad1`
`bi_flag`

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

Declared In
`OpenTransportProtocol.h`

boolean_p

```
typedef Boolean boolean_p;
```

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

Declared In
`OpenTransport.h`

caddr_t

```
typedef char * caddr_t;
```

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

Declared In
`types.h`

CCMiscInfo

```
struct CCMiscInfo {
    UInt32 connectionStatus;
    UInt32 connectionTimeElapsed;
    UInt32 connectionTimeRemaining;
    UInt32 bytesTransmitted;
    UInt32 bytesReceived;
    UInt32 reserved;
};
typedef struct CCMiscInfo CCMiscInfo;
```

Fields

`connectionStatus`

`connectionTimeElapsed`

`connectionTimeRemaining`

`bytesTransmitted`

`bytesReceived`

`reserved`

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

Declared In
`OpenTransportProviders.h`

CFMLibraryInfo

```
struct CFMLibraryInfo {
    OTLink link;
    char * libName;
    StringPtr intlName;
    FSSpec * fileSpec;
    StringPtr pstring2;
    StringPtr pstring3;
};
typedef struct CFMLibraryInfo CFMLibraryInfo;
```

Fields

`link`

`libName`

`intlName`

`fileSpec`

`pstring2`

`pstring3`

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

Declared In
`OpenTransportProtocol.h`

char_p

```
typedef SInt8 char_p;
```

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

Declared In
`OpenTransport.h`

copyreq

```
struct copyreq {
    SInt32 cq_cmd;
    cred * cq_cr;
    UInt32 cq_id;
    caddr_t cq_addr;
    UInt32 cq_size;
    SInt32 cq_flag;
    mblk_t * cq_private;
    long cq_filler[4];
};
typedef struct copyreq copyreq;
```

Fields
`cq_cmd`
`cq_cr`
`cq_id`
`cq_addr`
`cq_size`
`cq_flag`
`cq_private`
`cq_filler`

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

Declared In
`OpenTransportProtocol.h`

copyresp

```
struct copyresp {
    SInt32 cp_cmd;
    cred * cp_cr;
    UInt32 cp_id;
    caddr_t cp_rval;
    UInt32 cp_pad1;
    SInt32 cp_pad2;
    mblk_t * cp_private;
    long cp_filler[4];
};
typedef struct copyresp copyresp;
```

Fields

`cp_cmd`

`cp_cr`

`cp_id`

`cp_rval`

`cp_pad1`

`cp_pad2`

`cp_private`

`cp_filler`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

cred

```
struct cred {
    UInt16 cr_ref;
    UInt16 cr_ngroups;
    uid_t cr_uid;
    gid_t cr_gid;
    uid_t cr_ruid;
    gid_t cr_rgid;
    uid_t cr_suid;
    gid_t cr_sgid;
    gid_t cr_groups[1];
};
typedef struct cred cred;
typedef cred cred_t;
```

Fields

`cr_ref`

`cr_ngroups`

`cr_uid`

`cr_gid`

`cr_ruid`

`cr_rgid`

`cr_suid`

`cr_sgid`

`cr_groups`

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

Declared In
`OpenTransportProtocol.h`

cred_t

```
typedef cred cred_t;
```

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

Declared In
`OpenTransportProtocol.h`

datab

```
struct datab {
    datab_db_f db_f;
    unsigned char * db_base;
    unsigned char * db_lim;
    unsigned char db_ref;
    unsigned char db_type;
    unsigned char db_iswhat;
    unsigned char db_filler2;
    UInt32 db_size;
    unsigned char * db_msgaddr;
    long db_filler;
};
typedef struct datab datab;
typedef datab dblk_t;
```

Fields

`db_f`

`db_base`

`db_lim`

`db_ref`

`db_type`

`db_iswhat`

`db_filler2`

`db_size`

`db_msgaddr`

`db_filler`

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

Declared In
`OpenTransportProtocol.h`

datab_db_f

```
union datab_db_f {
    datab * freep;
    free_rtn * frtnp;
};
typedef union datab_db_f datab_db_f;
```

Fields

`freep`

`frtnp`

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

Declared In
`OpenTransportProtocol.h`

dblk_t

```
typedef datab dblk_t;
```

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

Declared In
`OpenTransportProtocol.h`

DDPAddress

```
struct DDPAddress {
    OTAddressType fAddressType;
    UInt16 fNetwork;
    UInt8 fNodeID;
    UInt8 fSocket;
    UInt8 fDDPType;
    UInt8 fPad;
};
typedef struct DDPAddress DDPAddress;
```

Fields
`fAddressType`
`fNetwork`
`fNodeID`
`fSocket`
`fDDPType`
`fPad`

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

Declared In
`OpenTransportProviders.h`

DDPNBPAddress

```
struct DDPNBPAddress {
    OTAddressType fAddressType;
    UInt16 fNetwork;
    UInt8 fNodeID;
    UInt8 fSocket;
    UInt8 fDDPType;
    UInt8 fPad;
    UInt8 fNBPNameBuffer[105];
};
typedef struct DDPNBPAddress DDPNBPAddress;
```

Fields

`fAddressType`

`fNetwork`

`fNodeID`

`fSocket`

`fDDPType`

`fPad`

`fNBPNameBuffer`

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

Declared In
`OpenTransportProviders.h`

dev_t

```
typedef UInt32 dev_t;
```

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

Declared In
`types.h`

dl_attach_req_t

```
struct dl_attach_req_t {
    UInt32 dl_primitive;
    UInt32 dl_ppa;
};
typedef struct dl_attach_req_t dl_attach_req_t;
```

Fields

`dl_primitive`

`dl_ppa`

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

Declared In
OpenTransportProtocol.h

dl_bind_ack_t

```
struct dl_bind_ack_t {
    UInt32 dl_primitive;
    UInt32 dl_sap;
    UInt32 dl_addr_length;
    UInt32 dl_addr_offset;
    UInt32 dl_max_conind;
    UInt32 dl_xidtest_flg;
};
typedef struct dl_bind_ack_t dl_bind_ack_t;
```

Fields
dl_primitive
dl_sap
dl_addr_length
dl_addr_offset
dl_max_conind
dl_xidtest_flg

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

Declared In
OpenTransportProtocol.h

dl_bind_req_t

```
struct dl_bind_req_t {
    UInt32 dl_primitive;
    UInt32 dl_sap;
    UInt32 dl_max_conind;
    UInt16 dl_service_mode;
    UInt16 dl_conn_mgmt;
    UInt32 dl_xidtest_flg;
};
typedef struct dl_bind_req_t dl_bind_req_t;
```

Fields
dl_primitive
dl_sap
dl_max_conind
dl_service_mode
dl_conn_mgmt
dl_xidtest_flg

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

Declared In
OpenTransportProtocol.h

dl_connect_con_t

```
struct dl_connect_con_t {
    UInt32 dl_primitive;
    UInt32 dl_resp_addr_length;
    UInt32 dl_resp_addr_offset;
    UInt32 dl_qos_length;
    UInt32 dl_qos_offset;
    UInt32 dl_growth;
};
typedef struct dl_connect_con_t dl_connect_con_t;
```

Fields
dl_primitive
dl_resp_addr_length
dl_resp_addr_offset
dl_qos_length
dl_qos_offset
dl_growth

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

Declared In
OpenTransportProtocol.h

dl_connect_ind_t

```
struct dl_connect_ind_t {
    UInt32 dl_primitive;
    UInt32 dl_correlation;
    UInt32 dl_called_addr_length;
    UInt32 dl_called_addr_offset;
    UInt32 dl_calling_addr_length;
    UInt32 dl_calling_addr_offset;
    UInt32 dl_qos_length;
    UInt32 dl_qos_offset;
    UInt32 dl_growth;
};
typedef struct dl_connect_ind_t dl_connect_ind_t;
```

Fields

`dl_primitive`

`dl_correlation`

`dl_called_addr_length`

`dl_called_addr_offset`

`dl_calling_addr_length`

`dl_calling_addr_offset`

`dl_qos_length`

`dl_qos_offset`

`dl_growth`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_connect_req_t

```
struct dl_connect_req_t {
    UInt32 dl_primitive;
    UInt32 dl_dest_addr_length;
    UInt32 dl_dest_addr_offset;
    UInt32 dl_qos_length;
    UInt32 dl_qos_offset;
    UInt32 dl_growth;
};
typedef struct dl_connect_req_t dl_connect_req_t;
```

Fields

`dl_primitive`

`dl_dest_addr_length`

`dl_dest_addr_offset`

`dl_qos_length`

`dl_qos_offset`

`dl_growth`

Availability

Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

dl_connect_res_t

```
struct dl_connect_res_t {
    UInt32 dl_primitive;
    UInt32 dl_correlation;
    UInt32 dl_resp_token;
    UInt32 dl_qos_length;
    UInt32 dl_qos_offset;
    UInt32 dl_growth;
};
typedef struct dl_connect_res_t dl_connect_res_t;
```

Fields
dl_primitive
dl_correlation
dl_resp_token
dl_qos_length
dl_qos_offset
dl_growth

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

Declared In
OpenTransportProtocol.h

dl_data_ack_ind_t

```
struct dl_data_ack_ind_t {
    UInt32 dl_primitive;
    UInt32 dl_dest_addr_length;
    UInt32 dl_dest_addr_offset;
    UInt32 dl_src_addr_length;
    UInt32 dl_src_addr_offset;
    UInt32 dl_priority;
    UInt32 dl_service_class;
};
typedef struct dl_data_ack_ind_t dl_data_ack_ind_t;
```

Fields
dl_primitive
dl_dest_addr_length
dl_dest_addr_offset
dl_src_addr_length
dl_src_addr_offset
dl_priority
dl_service_class

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

Declared In
OpenTransportProtocol.h

dl_data_ack_req_t

```
struct dl_data_ack_req_t {
    UInt32 dl_primitive;
    UInt32 dl_correlation;
    UInt32 dl_dest_addr_length;
    UInt32 dl_dest_addr_offset;
    UInt32 dl_src_addr_length;
    UInt32 dl_src_addr_offset;
    UInt32 dl_priority;
    UInt32 dl_service_class;
};
typedef struct dl_data_ack_req_t dl_data_ack_req_t;
```

Fields
dl_primitive
dl_correlation
dl_dest_addr_length
dl_dest_addr_offset
dl_src_addr_length
dl_src_addr_offset
dl_priority
dl_service_class

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

Declared In
OpenTransportProtocol.h

dl_data_ack_status_ind_t

```
struct dl_data_ack_status_ind_t {
    UInt32 dl_primitive;
    UInt32 dl_correlation;
    UInt32 dl_status;
};
typedef struct dl_data_ack_status_ind_t dl_data_ack_status_ind_t;
```

Fields
dl_primitive
dl_correlation
dl_status

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

Declared In
OpenTransportProtocol.h

dl_detach_req_t

```
struct dl_detach_req_t {
    UInt32 dl_primitive;
};
typedef struct dl_detach_req_t dl_detach_req_t;
```

Fields
`dl_primitive`

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

Declared In
`OpenTransportProtocol.h`

dl_disabmulti_req_t

```
struct dl_disabmulti_req_t {
    UInt32 dl_primitive;
    UInt32 dl_addr_length;
    UInt32 dl_addr_offset;
};
typedef struct dl_disabmulti_req_t dl_disabmulti_req_t;
```

Fields
`dl_primitive`
`dl_addr_length`
`dl_addr_offset`

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

Declared In
`OpenTransportProtocol.h`

dl_disconnect_ind_t

```
struct dl_disconnect_ind_t {
    UInt32 dl_primitive;
    UInt32 dl_originator;
    UInt32 dl_reason;
    UInt32 dl_correlation;
};
typedef struct dl_disconnect_ind_t dl_disconnect_ind_t;
```

Fields
`dl_primitive`
`dl_originator`
`dl_reason`
`dl_correlation`

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

Declared In
OpenTransportProtocol.h

dl_disconnect_req_t

```
struct dl_disconnect_req_t {
    UInt32 dl_primitive;
    UInt32 dl_reason;
    UInt32 dl_correlation;
};
typedef struct dl_disconnect_req_t dl_disconnect_req_t;
```

Fields
dl_primitive
dl_reason
dl_correlation

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

Declared In
OpenTransportProtocol.h

dl_enabmulti_req_t

```
struct dl_enabmulti_req_t {
    UInt32 dl_primitive;
    UInt32 dl_addr_length;
    UInt32 dl_addr_offset;
};
typedef struct dl_enabmulti_req_t dl_enabmulti_req_t;
```

Fields
dl_primitive
dl_addr_length
dl_addr_offset

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

Declared In
OpenTransportProtocol.h

dl_error_ack_t

```
struct dl_error_ack_t {
    UInt32 dl_primitive;
    UInt32 dl_error_primitive;
    UInt32 dl_errno;
    UInt32 dl_unix_errno;
};
typedef struct dl_error_ack_t dl_error_ack_t;
```

Fields

`dl_primitive`

`dl_error_primitive`

`dl_errno`

`dl_unix_errno`

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

Declared In
`OpenTransportProtocol.h`

dl_get_statistics_ack_t

```
struct dl_get_statistics_ack_t {
    UInt32 dl_primitive;
    UInt32 dl_stat_length;
    UInt32 dl_stat_offset;
};
typedef struct dl_get_statistics_ack_t dl_get_statistics_ack_t;
```

Fields

`dl_primitive`

`dl_stat_length`

`dl_stat_offset`

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

Declared In
`OpenTransportProtocol.h`

dl_get_statistics_req_t

```
struct dl_get_statistics_req_t {
    UInt32 dl_primitive;
};
typedef struct dl_get_statistics_req_t dl_get_statistics_req_t;
```

Fields

`dl_primitive`

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

Declared In
OpenTransportProtocol.h

dl_info_ack_t

```
struct dl_info_ack_t {
    UInt32 dl_primitive;
    UInt32 dl_max_sdu;
    UInt32 dl_min_sdu;
    UInt32 dl_addr_length;
    UInt32 dl_mac_type;
    UInt32 dl_reserved;
    UInt32 dl_current_state;
    SInt32 dl_sap_length;
    UInt32 dl_service_mode;
    UInt32 dl_qos_length;
    UInt32 dl_qos_offset;
    UInt32 dl_qos_range_length;
    UInt32 dl_qos_range_offset;
    UInt32 dl_provider_style;
    UInt32 dl_addr_offset;
    UInt32 dl_version;
    UInt32 dl_brdcst_addr_length;
    UInt32 dl_brdcst_addr_offset;
    UInt32 dl_growth;
};
typedef struct dl_info_ack_t dl_info_ack_t;
```

Fields
dl_primitive
dl_max_sdu
dl_min_sdu
dl_addr_length
dl_mac_type
dl_reserved
dl_current_state
dl_sap_length
dl_service_mode
dl_qos_length
dl_qos_offset
dl_qos_range_length
dl_qos_range_offset
dl_provider_style
dl_addr_offset
dl_version
dl_brdcst_addr_length
dl_brdcst_addr_offset
dl_growth

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

Declared In
OpenTransportProtocol.h

dl_info_req_t

```
struct dl_info_req_t {
    UInt32 dl_primitive;
};
typedef struct dl_info_req_t dl_info_req_t;
```

Fields
dl_primitive

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

Declared In
OpenTransportProtocol.h

dl_ok_ack_t

```
struct dl_ok_ack_t {
    UInt32 dl_primitive;
    UInt32 dl_correct_primitive;
};
typedef struct dl_ok_ack_t dl_ok_ack_t;
```

Fields
dl_primitive
dl_correct_primitive

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

Declared In
OpenTransportProtocol.h

dl_phys_addr_ack_t

```
struct dl_phys_addr_ack_t {
    UInt32 dl_primitive;
    UInt32 dl_addr_length;
    UInt32 dl_addr_offset;
};
typedef struct dl_phys_addr_ack_t dl_phys_addr_ack_t;
```

Fields
dl_primitive
dl_addr_length
dl_addr_offset

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

Declared In
OpenTransportProtocol.h

dl_phys_addr_req_t

```
struct dl_phys_addr_req_t {
    UInt32 dl_primitive;
    UInt32 dl_addr_type;
};
typedef struct dl_phys_addr_req_t dl_phys_addr_req_t;
```

Fields
dl_primitive
dl_addr_type

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

Declared In
OpenTransportProtocol.h

DL_primitives

```
union DL_primitives {
    UInt32 dl_primitive;
    dl_info_req_t info_req;
    dl_info_ack_t info_ack;
    dl_attach_req_t attach_req;
    dl_detach_req_t detach_req;
    dl_bind_req_t bind_req;
    dl_bind_ack_t bind_ack;
    dl_unbind_req_t unbind_req;
    dl_subs_bind_req_t subs_bind_req;
    dl_subs_bind_ack_t subs_bind_ack;
    dl_subs_unbind_req_t subs_unbind_req;
    dl_ok_ack_t ok_ack;
    dl_error_ack_t error_ack;
    dl_connect_req_t connect_req;
    dl_connect_ind_t connect_ind;
    dl_connect_res_t connect_res;
    dl_connect_con_t connect_con;
    dl_token_req_t token_req;
    dl_token_ack_t token_ack;
    dl_disconnect_req_t disconnect_req;
    dl_disconnect_ind_t disconnect_ind;
    dl_reset_req_t reset_req;
    dl_reset_ind_t reset_ind;
    dl_reset_res_t reset_res;
    dl_reset_con_t reset_con;
    dl_unitdata_req_t unitdata_req;
    dl_unitdata_ind_t unitdata_ind;
    dl_uderror_ind_t uderror_ind;
    dl_udqos_req_t udqos_req;
    dl_enabmulti_req_t enabmulti_req;
    dl_disabmulti_req_t disabmulti_req;
    dl_promiscon_req_t promiscon_req;
    dl_promiscoff_req_t promiscoff_req;
    dl_phys_addr_req_t physaddr_req;
    dl_phys_addr_ack_t physaddr_ack;
    dl_set_phys_addr_req_t set_physaddr_req;
    dl_get_statistics_req_t get_statistics_req;
    dl_get_statistics_ack_t get_statistics_ack;
    dl_test_req_t test_req;
    dl_test_ind_t test_ind;
    dl_test_res_t test_res;
    dl_test_con_t test_con;
    dl_xid_req_t xid_req;
    dl_xid_ind_t xid_ind;
    dl_xid_res_t xid_res;
    dl_xid_con_t xid_con;
    dl_data_ack_req_t data_ack_req;
    dl_data_ack_ind_t data_ack_ind;
    dl_data_ack_status_ind_t data_ack_status_ind;
    dl_reply_req_t reply_req;
    dl_reply_ind_t reply_ind;
    dl_reply_status_ind_t reply_status_ind;
    dl_reply_update_req_t reply_update_req;
    dl_reply_update_status_ind_t reply_update_status_ind;
};
```

```
typedef union DL_primitives DL_primitives;
```

Fields

```
dl_primitive
info_req
info_ack
attach_req
detach_req
bind_req
bind_ack
unbind_req
subs_bind_req
subs_bind_ack
subs_unbind_req
ok_ack
error_ack
connect_req
connect_ind
connect_res
connect_con
token_req
token_ack
disconnect_req
disconnect_ind
reset_req
reset_ind
reset_res
reset_con
unitdata_req
unitdata_ind
uderror_ind
udqos_req
enabmulti_req
disabmulti_req
promiscon_req
promiscoff_req
physaddr_req
physaddr_ack
set_physaddr_req
get_statistics_req
get_statistics_ack
test_req
test_ind
test_res
test_con
xid_req
xid_ind
xid_res
xid_con
```

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

Declared In
`OpenTransportProtocol.h`

dl_priority_t

```
struct dl_priority_t {
    SInt32 dl_min;
    SInt32 dl_max;
};
typedef struct dl_priority_t dl_priority_t;
```

Fields
`dl_min`
`dl_max`

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

Declared In
`OpenTransportProtocol.h`

dl_promiscoff_req_t

```
struct dl_promiscoff_req_t {
    UInt32 dl_primitive;
    UInt32 dl_level;
};
typedef struct dl_promiscoff_req_t dl_promiscoff_req_t;
```

Fields
`dl_primitive`
`dl_level`

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

Declared In
`OpenTransportProtocol.h`

dl_promiscon_req_t

```
struct dl_promiscon_req_t {
    UInt32 dl_primitive;
    UInt32 dl_level;
};
typedef struct dl_promiscon_req_t dl_promiscon_req_t;
```

Fields

`dl_primitive`

`dl_level`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_protect_t

```
struct dl_protect_t {
    SInt32 dl_min;
    SInt32 dl_max;
};
typedef struct dl_protect_t dl_protect_t;
```

Fields

`dl_min`

`dl_max`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_qos_cl_range1_t

```
struct dl_qos_cl_range1_t {
    UInt32 dl_qos_type;
    dl_transdelay_t dl_trans_delay;
    dl_priority_t dl_priority;
    dl_protect_t dl_protection;
    SInt32 dl_residual_error;
};
typedef struct dl_qos_cl_range1_t dl_qos_cl_range1_t;
```

Fields

`dl_qos_type`

`dl_trans_delay`

`dl_priority`

`dl_protection`

`dl_residual_error`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_qos_cl_sel1_t

```
struct dl_qos_cl_sel1_t {
    UInt32 dl_qos_type;
    SInt32 dl_trans_delay;
    SInt32 dl_priority;
    SInt32 dl_protection;
    SInt32 dl_residual_error;
};
typedef struct dl_qos_cl_sel1_t dl_qos_cl_sel1_t;
```

Fields

`dl_qos_type`

`dl_trans_delay`

`dl_priority`

`dl_protection`

`dl_residual_error`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_qos_co_range1_t

```
struct dl_qos_co_range1_t {
    UInt32 dl_qos_type;
    dl_through_t dl_rcv_throughput;
    dl_transdelay_t dl_rcv_trans_delay;
    dl_through_t dl_xmt_throughput;
    dl_transdelay_t dl_xmt_trans_delay;
    dl_priority_t dl_priority;
    dl_protect_t dl_protection;
    SInt32 dl_residual_error;
    dl_resilience_t dl_resilience;
};
typedef struct dl_qos_co_range1_t dl_qos_co_range1_t;
```

Fields

`dl_qos_type`

`dl_rcv_throughput`

`dl_rcv_trans_delay`

`dl_xmt_throughput`

`dl_xmt_trans_delay`

`dl_priority`

`dl_protection`

`dl_residual_error`

`dl_resilience`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_qos_co_sel1_t

```
struct dl_qos_co_sel1_t {
    UInt32 dl_qos_type;
    SInt32 dl_rcv_throughput;
    SInt32 dl_rcv_trans_delay;
    SInt32 dl_xmt_throughput;
    SInt32 dl_xmt_trans_delay;
    SInt32 dl_priority;
    SInt32 dl_protection;
    SInt32 dl_residual_error;
    dl_resilience_t dl_resilience;
};
typedef struct dl_qos_co_sel1_t dl_qos_co_sel1_t;
```

Fields

`dl_qos_type`
`dl_rcv_throughput`
`dl_rcv_trans_delay`
`dl_xmt_throughput`
`dl_xmt_trans_delay`
`dl_priority`
`dl_protection`
`dl_residual_error`
`dl_resilience`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_reply_ind_t

```
struct dl_reply_ind_t {
    UInt32 dl_primitive;
    UInt32 dl_dest_addr_length;
    UInt32 dl_dest_addr_offset;
    UInt32 dl_src_addr_length;
    UInt32 dl_src_addr_offset;
    UInt32 dl_priority;
    UInt32 dl_service_class;
};
typedef struct dl_reply_ind_t dl_reply_ind_t;
```

Fields

`dl_primitive`

`dl_dest_addr_length`

`dl_dest_addr_offset`

`dl_src_addr_length`

`dl_src_addr_offset`

`dl_priority`

`dl_service_class`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_reply_req_t

```
struct dl_reply_req_t {
    UInt32 dl_primitive;
    UInt32 dl_correlation;
    UInt32 dl_dest_addr_length;
    UInt32 dl_dest_addr_offset;
    UInt32 dl_src_addr_length;
    UInt32 dl_src_addr_offset;
    UInt32 dl_priority;
    UInt32 dl_service_class;
};
typedef struct dl_reply_req_t dl_reply_req_t;
```

Fields

`dl_primitive`

`dl_correlation`

`dl_dest_addr_length`

`dl_dest_addr_offset`

`dl_src_addr_length`

`dl_src_addr_offset`

`dl_priority`

`dl_service_class`

Availability

Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

dl_reply_status_ind_t

```
struct dl_reply_status_ind_t {
    UInt32 dl_primitive;
    UInt32 dl_correlation;
    UInt32 dl_status;
};
typedef struct dl_reply_status_ind_t dl_reply_status_ind_t;
```

Fields
dl_primitive
dl_correlation
dl_status

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

Declared In
OpenTransportProtocol.h

dl_reply_update_req_t

```
struct dl_reply_update_req_t {
    UInt32 dl_primitive;
    UInt32 dl_correlation;
    UInt32 dl_src_addr_length;
    UInt32 dl_src_addr_offset;
};
typedef struct dl_reply_update_req_t dl_reply_update_req_t;
```

Fields
dl_primitive
dl_correlation
dl_src_addr_length
dl_src_addr_offset

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

Declared In
OpenTransportProtocol.h

dl_reply_update_status_ind_t

```
struct dl_reply_update_status_ind_t {
    UInt32 dl_primitive;
    UInt32 dl_correlation;
    UInt32 dl_status;
};
typedef struct dl_reply_update_status_ind_t dl_reply_update_status_ind_t;
```

Fields

`dl_primitive`

`dl_correlation`

`dl_status`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_reset_con_t

```
struct dl_reset_con_t {
    UInt32 dl_primitive;
};
typedef struct dl_reset_con_t dl_reset_con_t;
```

Fields

`dl_primitive`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_reset_ind_t

```
struct dl_reset_ind_t {
    UInt32 dl_primitive;
    UInt32 dl_originator;
    UInt32 dl_reason;
};
typedef struct dl_reset_ind_t dl_reset_ind_t;
```

Fields

`dl_primitive`

`dl_originator`

`dl_reason`

Availability

Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

dl_reset_req_t

```
struct dl_reset_req_t {
    UInt32 dl_primitive;
};
typedef struct dl_reset_req_t dl_reset_req_t;
```

Fields
dl_primitive

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

Declared In
OpenTransportProtocol.h

dl_reset_res_t

```
struct dl_reset_res_t {
    UInt32 dl_primitive;
};
typedef struct dl_reset_res_t dl_reset_res_t;
```

Fields
dl_primitive

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

Declared In
OpenTransportProtocol.h

dl_resilience_t

```
struct dl_resilience_t {
    SInt32 dl_disc_prob;
    SInt32 dl_reset_prob;
};
typedef struct dl_resilience_t dl_resilience_t;
```

Fields
dl_disc_prob
dl_reset_prob

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

Declared In
OpenTransportProtocol.h

dl_set_phys_addr_req_t

```
struct dl_set_phys_addr_req_t {
    UInt32 dl_primitive;
    UInt32 dl_addr_length;
    UInt32 dl_addr_offset;
};
typedef struct dl_set_phys_addr_req_t dl_set_phys_addr_req_t;
```

Fields

`dl_primitive`

`dl_addr_length`

`dl_addr_offset`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_subs_bind_ack_t

```
struct dl_subs_bind_ack_t {
    UInt32 dl_primitive;
    UInt32 dl_subs_sap_offset;
    UInt32 dl_subs_sap_length;
};
typedef struct dl_subs_bind_ack_t dl_subs_bind_ack_t;
```

Fields

`dl_primitive`

`dl_subs_sap_offset`

`dl_subs_sap_length`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_subs_bind_req_t

```
struct dl_subs_bind_req_t {
    UInt32 dl_primitive;
    UInt32 dl_subs_sap_offset;
    UInt32 dl_subs_sap_length;
    UInt32 dl_subs_bind_class;
};
typedef struct dl_subs_bind_req_t dl_subs_bind_req_t;
```

Fields

`dl_primitive`

`dl_subs_sap_offset`

`dl_subs_sap_length`

`dl_subs_bind_class`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_subs_unbind_req_t

```
struct dl_subs_unbind_req_t {
    UInt32 dl_primitive;
    UInt32 dl_subs_sap_offset;
    UInt32 dl_subs_sap_length;
};
typedef struct dl_subs_unbind_req_t dl_subs_unbind_req_t;
```

Fields

`dl_primitive`

`dl_subs_sap_offset`

`dl_subs_sap_length`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_test_con_t

```
struct dl_test_con_t {
    UInt32 dl_primitive;
    UInt32 dl_flag;
    UInt32 dl_dest_addr_length;
    UInt32 dl_dest_addr_offset;
    UInt32 dl_src_addr_length;
    UInt32 dl_src_addr_offset;
};
typedef struct dl_test_con_t dl_test_con_t;
```

Fields

`dl_primitive`

`dl_flag`

`dl_dest_addr_length`

`dl_dest_addr_offset`

`dl_src_addr_length`

`dl_src_addr_offset`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_test_ind_t

```
struct dl_test_ind_t {
    UInt32 dl_primitive;
    UInt32 dl_flag;
    UInt32 dl_dest_addr_length;
    UInt32 dl_dest_addr_offset;
    UInt32 dl_src_addr_length;
    UInt32 dl_src_addr_offset;
};
typedef struct dl_test_ind_t dl_test_ind_t;
```

Fields

`dl_primitive`

`dl_flag`

`dl_dest_addr_length`

`dl_dest_addr_offset`

`dl_src_addr_length`

`dl_src_addr_offset`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_test_req_t

```
struct dl_test_req_t {
    UInt32 dl_primitive;
    UInt32 dl_flag;
    UInt32 dl_dest_addr_length;
    UInt32 dl_dest_addr_offset;
};
typedef struct dl_test_req_t dl_test_req_t;
```

Fields

`dl_primitive`

`dl_flag`

`dl_dest_addr_length`

`dl_dest_addr_offset`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_test_res_t

```
struct dl_test_res_t {
    UInt32 dl_primitive;
    UInt32 dl_flag;
    UInt32 dl_dest_addr_length;
    UInt32 dl_dest_addr_offset;
};
typedef struct dl_test_res_t dl_test_res_t;
```

Fields

`dl_primitive`

`dl_flag`

`dl_dest_addr_length`

`dl_dest_addr_offset`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_through_t

```
struct dl_through_t {
    SInt32 dl_target_value;
    SInt32 dl_accept_value;
};
typedef struct dl_through_t dl_through_t;
```

Fields

`dl_target_value`

`dl_accept_value`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_token_ack_t

```
struct dl_token_ack_t {
    UInt32 dl_primitive;
    UInt32 dl_token;
};
typedef struct dl_token_ack_t dl_token_ack_t;
```

Fields

`dl_primitive`

`dl_token`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_token_req_t

```
struct dl_token_req_t {
    UInt32 dl_primitive;
};
typedef struct dl_token_req_t dl_token_req_t;
```

Fields

`dl_primitive`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_transdelay_t

```
struct dl_transdelay_t {
    SInt32 dl_target_value;
    SInt32 dl_accept_value;
};
typedef struct dl_transdelay_t dl_transdelay_t;
```

Fields

`dl_target_value`

`dl_accept_value`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_uderror_ind_t

```
struct dl_uderror_ind_t {
    UInt32 dl_primitive;
    UInt32 dl_dest_addr_length;
    UInt32 dl_dest_addr_offset;
    UInt32 dl_unix_errno;
    UInt32 dl_errno;
};
typedef struct dl_uderror_ind_t dl_uderror_ind_t;
```

Fields

`dl_primitive`

`dl_dest_addr_length`

`dl_dest_addr_offset`

`dl_unix_errno`

`dl_errno`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_udqos_req_t

```
struct dl_udqos_req_t {
    UInt32 dl_primitive;
    UInt32 dl_qos_length;
    UInt32 dl_qos_offset;
};
typedef struct dl_udqos_req_t dl_udqos_req_t;
```

Fields

`dl_primitive`

`dl_qos_length`

`dl_qos_offset`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_unbind_req_t

```
struct dl_unbind_req_t {
    UInt32 dl_primitive;
};
typedef struct dl_unbind_req_t dl_unbind_req_t;
```

Fields

`dl_primitive`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_unitdata_ind_t

```
struct dl_unitdata_ind_t {
    UInt32 dl_primitive;
    UInt32 dl_dest_addr_length;
    UInt32 dl_dest_addr_offset;
    UInt32 dl_src_addr_length;
    UInt32 dl_src_addr_offset;
    UInt32 dl_group_address;
};
typedef struct dl_unitdata_ind_t dl_unitdata_ind_t;
```

Fields

`dl_primitive`

`dl_dest_addr_length`

`dl_dest_addr_offset`

`dl_src_addr_length`

`dl_src_addr_offset`

`dl_group_address`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_unitdata_req_t

```
struct dl_unitdata_req_t {
    UInt32 dl_primitive;
    UInt32 dl_dest_addr_length;
    UInt32 dl_dest_addr_offset;
    dl_priority_t dl_priority;
};
typedef struct dl_unitdata_req_t dl_unitdata_req_t;
```

Fields

`dl_primitive`

`dl_dest_addr_length`

`dl_dest_addr_offset`

`dl_priority`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_xid_con_t

```
struct dl_xid_con_t {
    UInt32 dl_primitive;
    UInt32 dl_flag;
    UInt32 dl_dest_addr_length;
    UInt32 dl_dest_addr_offset;
    UInt32 dl_src_addr_length;
    UInt32 dl_src_addr_offset;
};
typedef struct dl_xid_con_t dl_xid_con_t;
```

Fields

`dl_primitive`

`dl_flag`

`dl_dest_addr_length`

`dl_dest_addr_offset`

`dl_src_addr_length`

`dl_src_addr_offset`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_xid_ind_t

```
struct dl_xid_ind_t {
    UInt32 dl_primitive;
    UInt32 dl_flag;
    UInt32 dl_dest_addr_length;
    UInt32 dl_dest_addr_offset;
    UInt32 dl_src_addr_length;
    UInt32 dl_src_addr_offset;
};
typedef struct dl_xid_ind_t dl_xid_ind_t;
```

Fields

`dl_primitive`

`dl_flag`

`dl_dest_addr_length`

`dl_dest_addr_offset`

`dl_src_addr_length`

`dl_src_addr_offset`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

dl_xid_req_t

```
struct dl_xid_req_t {
    UInt32 dl_primitive;
    UInt32 dl_flag;
    UInt32 dl_dest_addr_length;
    UInt32 dl_dest_addr_offset;
};
typedef struct dl_xid_req_t dl_xid_req_t;
```

Fields

`dl_primitive`

`dl_flag`

`dl_dest_addr_length`

`dl_dest_addr_offset`

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

Declared In
`OpenTransportProtocol.h`

dl_xid_res_t

```
struct dl_xid_res_t {
    UInt32 dl_primitive;
    UInt32 dl_flag;
    UInt32 dl_dest_addr_length;
    UInt32 dl_dest_addr_offset;
};
typedef struct dl_xid_res_t dl_xid_res_t;
```

Fields

`dl_primitive`

`dl_flag`

`dl_dest_addr_length`

`dl_dest_addr_offset`

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

Declared In
`OpenTransportProtocol.h`

DNS Address Structure

Holds host names, partially or fully-qualified domain names, or dotted-decimal format Internet addresses for use with a variety of Open Transport functions.

```
struct DNSAddress {
    OTAddressType fAddressType;
    InetDomainName fName;
};
typedef struct DNSAddress DNSAddress;
```

Fields

`fAddressType`

The address type. For a DNSAddress structure, this should be AF_DNS.

`fName`

The name to be resolved by the DNR.

Discussion

You can use the DNS (domain name system) address structure with the OTConnect function (TCP), with the OTSndUData function (UDP), or with the OTResolveAddress function (either TCP or UDP). If you do so, TCP/IP will resolve the name for you automatically. You can use the OTInitDNSAddress function to fill in a DNS address structure. The DNS address structure is defined by the DNSAddress data type.

The address you specify can be just the host name (“otteam”), a partially qualified domain name (“otteam.ssw”), a fully qualified domain name (“otteam.ssw.apple.com.”), or an internet address in dotted-decimal format (“17.202.99.99”), and can optionally include a port number (“otteam.ssw.apple.com:25” or “17.202.99.99:25”).

Because the port number is not actually part of the domain name, it is possible to have a domain name–port number combination that exceeds 255 bytes. If you wish to specify such a string, you must provide a structure based on the DNS address structure that has sufficient space to contain the full string. In any case, the domain name itself cannot exceed 255 bytes.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProviders.h`

DNS Query Information Structure

Returns answers to DNS queries.

```
struct DNSQueryInfo {
    UInt16 qType;
    UInt16 qClass;
    UInt32 ttl;
    InetDomainName name;
    UInt16 responseType;
    UInt16 resourceLen;
    char resourceData[4];
};
typedef struct DNSQueryInfo DNSQueryInfo;
```

Fields

`qType`

The numerical value of the DNS resource record type, such as MX and PTR, for which you wish to query.

`qClass`

The numerical value of the DNS record class, such as Inet and Hesio, for which you wish to query.

`ttl`
: An integer indicating the DNS resource record's time to live (in seconds).

`name`
: The fully qualified domain name or address for which you made the query.

`responseType`
: The type of response.

`resourceLen`
: The actual length of the resource data returned.

`resourceData`
: The resource data that is returned. This is at least 4 bytes long, and is usually longer.

Discussion
The DNS query information structure is used by the TCP/IP service provider to return answers to DNS queries made using the OTInetQuery function. The DNS query information structure is defined by the DNSQueryInfo data type. For additional information about the constant values for the DNSQueryInfo fields, see the DNS Requests for Comments (RFCs), available over the World Wide Web.

See the Internet Standard for a definitive list of values for the qType, qClass, and respnoseType fields, and for a definition of the format of the resource data.

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

Declared In
`OpenTransportProviders.h`

EndpointRef

```
typedef struct OpaqueEndpointRef * EndpointRef;
```

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

Declared In
`OpenTransport.h`

EnetPacketHeader

```
struct EnetPacketHeader {
    UInt8 fDestAddr[6];
    UInt8 fSourceAddr[6];
    UInt16 fProto;
};
typedef struct EnetPacketHeader EnetPacketHeader;
```

Fields

`fDestAddr`

`fSourceAddr`

`fProto`

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

Declared In
`OpenTransportProviders.h`

free_rtn

```
struct free_rtn {
    FreeFuncType free_func;
    char * free_arg;
};
typedef struct free_rtn free_rtn;
typedef free_rtn frtn_t;
```

Fields

`free_func`

`free_arg`

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

Declared In
`OpenTransportProtocol.h`

frtn_t

```
typedef free_rtn frtn_t;
```

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

Declared In
`OpenTransportProtocol.h`

gid_t

```
typedef UInt32 gid_t;
```

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

Declared In
`types.h`

Internet Address Structure

Used for providing a TCP or UDP address to the `OTConnect`, `OTSndURequest`, or `OTBind` functions.

```
struct InetAddress {
    OTAddressType fAddressType;
    InetPort fPort;
    InetHost fHost;
    UInt8 fUnused[8];
};
typedef struct InetAddress InetAddress;
```

Fields

`fAddressType`
The address type. The field should be AF_INET, which identifies the structure as an InetAddress.

`fPort`
The port number.

`fHost`
The 32-bit IP address of the host.

`fUnused`
Reserved.

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

Declared In
`OpenTransportProviders.h`

InetDHCPOption

```
struct InetDHCPOption {
    UInt8 fOptionTag;
    UInt8 fOptionLen;
    UInt8 fOptionValue;
};
typedef struct InetDHCPOption InetDHCPOption;
```

Fields
`fOptionTag`
`fOptionLen`
`fOptionValue`

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

Declared In
`OpenTransportProviders.h`

InetDomainName

```
typedef  InetDomainName[256];
```

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

Declared In
`OpenTransportProviders.h`

InetHost

```
typedef UInt32 InetHost;
```

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

Declared In
`OpenTransportProviders.h`

Internet Host Information Sructure

Holds a set of IP addresses associated with an Internet host for use by the `OTInetStringToAddress` function.

```
struct InetHostInfo {
    InetDomainName name;
    InetHost addrs[10];
};
typedef struct InetHostInfo InetHostInfo;
```

Fields

`name`

The canonical name of the host. The canonical name is a fully qualified domain nam, never an alias.

`addrs`

Up to ten IP addresses associated with this host name. Only multihomed hosts have more than one IP address.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProviders.h`

Internet Interface Information Structure

Holds information about an Internet interface for use by the `OTInetGetInterfaceInfo` function.

```
struct InetInterfaceInfo {
    InetHost fAddress;
    InetHost fNetmask;
    InetHost fBroadcastAddr;
    InetHost fDefaultGatewayAddr;
    InetHost fDNSAddr;
    UInt16 fVersion;
    UInt16 fHWAddrLen;
    UInt8 * fHWAddr;
    UInt32 fIfMTU;
    UInt8 * fReservedPtrs[2];
    InetDomainName fDomainName;
    UInt32 fIPSecondaryCount;
    UInt8 fReserved[252];
};
typedef struct InetInterfaceInfo InetInterfaceInfo;
```

Fields

`fAddress`

The IP address of the interface.

`fNetmask`

The subnet mask of the local IP network.

`fBroadcastAddr`

The broadcast address for the interface.

`fDefaultGatewayAddr`

The IP address of the default router. The default is a router that can forward any packet destined outside the locally connected subnet.

`fDNSAddr`

The address of a domain name server. This value can be returned by a server or typed in by the user during configuration of the TCP/IP interface.

`fVersion`
The version of the OTInetGetInterfaceInfo function; currently equal to kInetInterfaceInfoVersion.

`fHWAddrLen`
The length (in bytes) of the hardware address. This points into the fReserved field of this structure. It can be nil if the interface has no hardware address or if it won't fit.

`fHWAddr`
A pointer to the hardware address.

`fIfMTU`
The maximum transmission unit size in bytes permitted for this interface's hardware.

`fReservedPtrs`
Reserved.

`fDomainName`
The default domain name of the host as configured in the TCP/IP control panel. This name doesn't include the host name.

`fIPSecondaryCount`
The number of IP secondary address available.

`fReserved`
Reserved.

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

Declared In
`OpenTransportProviders.h`

Internet Mail Exchange Structure

Holds host names and mail preference values for use with the `OTInetMailExchange` function.

```
struct InetMailExchange {
    UInt16 preference;
    InetDomainName exchange;
};
typedef struct InetMailExchange InetMailExchange;
```

Fields
`preference`
The mail exchange preference value. The mail exchanger with the lowest preference number is the first one to which mail should be sent.

`exchange`
The fully qualified domain name of a host that can accept mail for your target host.

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

Declared In
`OpenTransportProviders.h`

InetPort

```
typedef UInt16 InetPort;
```

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

Declared In
`OpenTransportProviders.h`

InetSvcRef

```
typedef struct OpaqueInetSvcRef * InetSvcRef;
```

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

Declared In
`OpenTransportProviders.h`

InetSysInfo

Holds information about an Internet host for use by the `OTInetSysInfo` function.

```
struct InetSysInfo {
    char cpuType[32];
    char osType[32];
};
typedef struct InetSysInfo InetSysInfo;
```

Fields

`cpuType`
The CPU type of the specified host. This is an ASCII string maintained by the domain name server.

`osType`
The operating system running on the specified host. This is an ASCII string maintained by the domain name server.

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

Declared In
`OpenTransportProviders.h`

install_info

Fields

int_t

```
typedef  int_t;
```

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

Declared In
`OpenTransport.h`

iocblk

```
struct iocblk {
    SInt32 ioc_cmd;
    cred * ioc_cr;
    UInt32 ioc_id;
    UInt32 ioc_count;
    SInt32 ioc_error;
    SInt32 ioc_rval;
    long ioc_filler[4];
};
typedef struct iocblk iocblk;
```

Fields
`ioc_cmd`
`ioc_cr`
`ioc_id`
`ioc_count`
`ioc_error`
`ioc_rval`
`ioc_filler`

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

Declared In
`OpenTransportProtocol.h`

LCPEcho

```
struct LCPEcho {
    UInt32 retryCount;
    UInt32 retryPeriod;
};
typedef struct LCPEcho LCPEcho;
```

Fields

`retryCount`

`retryPeriod`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProviders.h`

linkblk

```
struct linkblk {
    queue_t * l_qtop;
    queue_t * l_qbot;
    SInt32 l_index;
    long l_pad[5];
};
typedef struct linkblk linkblk;
```

Fields

`l_qtop`

`l_qbot`

`l_index`

`l_pad`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

log_ctl

```
struct log_ctl {
    short mid;
    short sid;
    char level;
    char pad1;
    short flags;
    long ltime;
    long ttime;
    SInt32 seq_no;
};
typedef struct log_ctl log_ctl;
```

Fields

`mid`

`sid`

`level`

`pad1`

`flags`

`ltime`

`ttime`

`seq_no`

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

Declared In
`OpenTransportProtocol.h`

major_t

```
typedef UInt16 major_t;
```

MapperRef

```
typedef struct OpaqueMapperRef * MapperRef;
```

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

Declared In
`OpenTransport.h`

mblk_t

```
typedef msgb mblk_t;
```

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

Declared In
OpenTransportProtocol.h

minor_t

```
typedef UInt16 minor_t;
```

module_info

```
struct module_info {
    unsigned short mi_idnum;
    char * mi_idname;
    long mi_minpsz;
    long mi_maxpsz;
    unsigned long mi_hiwat;
    unsigned long mi_lowat;
};
typedef struct module_info module_info;
typedef module_info * module_infoPtr;
```

Fields
mi_idnum
mi_idname
mi_minpsz
mi_maxpsz
mi_hiwat
mi_lowat

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

Declared In
OpenTransportProtocol.h

module_stat

```
struct module_stat {
    long ms_pcnt;
    long ms_scnt;
    long ms_ocnt;
    long ms_ccnt;
    long ms_acnt;
    char * ms_xptr;
    short ms_xsize;
};
typedef struct module_stat module_stat;
```

Fields

`ms_pcnt`

`ms_scnt`

`ms_ocnt`

`ms_ccnt`

`ms_acnt`

`ms_xptr`

`ms_xsize`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

MPS_INTR_STATE

```
typedef UInt8 MPS_INTR_STATE;
```

msgb

```
struct msgb {
    struct msgb * b_next;
    struct msgb * b_prev;
    struct msgb * b_cont;
    unsigned char * b_rptr;
    unsigned char * b_wptr;
    datab * b_datap;
    unsigned char b_band;
    unsigned char b_pad1;
    unsigned short b_flag;
};
typedef struct msgb msgb;
typedef msgb mblk_t;
```

Fields

`b_next`

`b_prev`

`b_cont`

`b_rptr`

`b_wptr`

`b_datap`

`b_band`

`b_pad1`

`b_flag`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

NBPAddress

```
struct NBPAddress {
    OTAddressType fAddressType;
    UInt8 fNBPNameBuffer[105];
};
typedef struct NBPAddress NBPAddress;
```

Fields

`fAddressType`

`fNBPNameBuffer`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProviders.h`

NBPEntity

```
struct NBPEntity {
    UInt8 fEntity[99];
};
typedef struct NBPEntity NBPEntity;
```

Fields
`fEntity`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProviders.h`

netbuf

```
struct netbuf {
     maxlen;
     len;
    char *buf;
};
```

Fields

ot_bind

Fields

ot_optmgmt

Fields

OTAddress

Defines the common structure for all Open Transport addresses.

```
struct OTAddress {
    OTAddressType fAddressType;
    UInt8 fAddress[1];
};
typedef struct OTAddress OTAddress;
```

Fields

`fAddressType`

`fAddress`

Discussion

The `OTAddress` type itself is abstract. You would not declare a structure of this type because it does not contain any address information. However, address formats defined by Open Transport protocols all use the `fAddressType` field to describe the format of the fields to follow, which do contain address information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransport.h`

OTAddressType

```
typedef UInt16 OTAddressType;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransport.h`

OTAutopushInfo

```
struct OTAutopushInfo {
    UInt32 sap_cmd;
    char sap_device_name[32];
    SInt32 sap_minor;
    SInt32 sap_lastminor;
    SInt32 sap_npush;
    char sap_list[8][32];
};
typedef struct OTAutopushInfo OTAutopushInfo;
```

Fields

`sap_cmd`

`sap_device_name`

`sap_minor`

`sap_lastminor`

`sap_npush`

`sap_list`

Availability

Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

OTBand

```
typedef UInt32 OTBand;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTBooleanParam

```
typedef Boolean OTBooleanParam;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

No-Copy Receive Buffer Structure

Receives data without copying it.

```
struct OTBuffer {
    void * fLink;
    void * fLink2;
    OTBuffer * fNext;
    UInt8 * fData;
    ByteCount fLen;
    void * fSave;
    UInt8 fBand;
    UInt8 fType;
    UInt8 fPad1;
    UInt8 fFlags;
};
typedef struct OTBuffer OTBuffer;
```

Fields

`fLink`
Reserved.

`fLink2`
Reserved.

`fNext`
A pointer to the next OTBuffer structure in the linked chain. By tracing the chain of fNext pointers, you can access all of the data associated with the message.

`fData`
: A pointer to the data portion of this OTBuffer structure.

`fLen`
: The length of data pointed to by the fData field.

`fSave`
: Reserved.

`fBand`
: The band used for the data transmission. It must be a value between 0 and 255.

`fType`
: The type of the data (normally M_DATA, M_PROTO, or M_PCPROTO).

`fPad1`
: Reserved.

`fFlags`
: The flags associated with the data (MSGMARK, MSGDELIM).

Discussion
You use the no-copy receive buffer structure when you wish to receive data without copying it with the OTRcvUData function, the OTRcvURequest function, the OTRcvUReply function, the OTRcv function, the OTRcvRequest function, and the OTRcvReply function.

If you are familiar with STREAMS mblk_t data structures, you can see that the no-copy receive buffer structure is just a slight modification of the mblk_t structure.

You can only use this buffer for data; you cannot use it for the address or options that may be associated with the incoming data. For example, in the case of an incoming TUnitData structure, you can only no-copy receive the udata portion, not the addr or opt fields.

Special Considerations

Under no circumstance write to this data structure. It is read-only. If you write to it, you can crash the system.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

Buffer Information Structure

A convenience structure for keeping track of where your application left off in an OTBuffer structure.

```
struct OTBufferInfo {
    OTBuffer * fBuffer;
    ByteCount fOffset;
    UInt8 fPad;
};
typedef struct OTBufferInfo OTBufferInfo;
```

Fields
`fBuffer`
: A pointer to the no-copy receive buffer.

`fOffset`
An offset indicating how far into the buffer you have read.

`fPad`
Reserved.

Discussion
The buffer information structure is provided for your convenience in keeping track of where you last left off in an OTBuffer structure. Because the no-copy receive buffer structure (OTBuffer) is read-only, you may need to copy the data in sections as you progress through the no-copy receive buffer. The utility function OTReadBuffer is used with this structure to easily copy the data out of an OTBuffer structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

OTByteCount

```
typedef ByteCount OTByteCount;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

OTClient

```
typedef struct OpaqueOTClient * OTClient;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

OTClientContextPtr

```
typedef struct OpaqueOTClientContextPtr * OTClientContextPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

OTClientList

Identifies the clients that denied a request to yield a port.

```
struct OTClientList {
    ItemCount fNumClients;
    UInt8 fBuffer[4];
};
typedef struct OTClientList OTClientList;
```

Fields

`fNumClients`

The number of clients in the fBuffer array, normally 1.

`fBuffer`

An array of packed Pascal strings enumerating the name of each client that rejected the request—that is, the names under which the clients registered themselves as an Open Transport clients.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

OTClientName

```
typedef UInt8 * OTClientName;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

OTCommand

```
typedef SInt32 OTCommand;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

OTConfigurationRef

```
typedef struct OTConfiguration * OTConfigurationRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

OTData Structure

Specifies the location and size of noncontiguous data.

```
struct OTData {
    void * fNext;
    void * fData;
    ByteCount fLen;
};
typedef struct OTData OTData;
```

Fields

`fNext`

A pointer to the OTData structure that describes the next data fragment. Specify a NULL pointer for the last data fragment.

`fData`

A pointer to the data fragment.

`fLen`

Specifies the size of the fragment in bytes.

Discussion

The OTData structure is an Apple extension used to specify the location and size of noncontiguous data. You use a pointer to this structure in place of a pointer to continguous data normally referenced in TNetbuf.buf field. You can send discontiguous data using the OTSndUData function, the OTSndURequest function, the OTSndUReply function, the OTSnd function, the OTSndRequest function, and the OTSndReply function.

Each OTData structure specifies the location of a data fragment, the size of the fragment, and the location of the OTData structure that specifies the location and size of the next data fragment. The data information structure is defined by the OTData type. For more information, see "Sending Noncontiguous Data."

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransport.h`

OTDataSize

```
typedef SInt32 OTDataSize;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransport.h`

OTDeferredTaskRef

```
typedef long OTDeferredTaskRef;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTEventCode

```
typedef UInt32 OTEventCode;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTError

```
typedef SInt32 OTError;
```

OTGate

```
struct OTGate {
    OTLIFO fLIFO;
    OTList fList;
    OTGateProcPtr fProc;
    SInt32 fNumQueued;
    SInt32 fInside;
};
typedef struct OTGate OTGate;
```

Fields
fLIFO
fList
fProc
fNumQueued
fInside

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

OTHashList

```
struct OTHashList {
    OTHashProcPtr fHashProc;
    ByteCount fHashTableSize;
    OTLink ** fHashBuckets;
};
typedef struct OTHashList OTHashList;
```

Fields

`fHashProc`

`fHashTableSize`

`fHashBuckets`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

OTInt32

```
typedef SInt32 OTInt32;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

OTISDNAddress

```
struct OTISDNAddress {
    OTAddressType fAddressType;
    UInt16 fPhoneLength;
    char fPhoneNumber[37];
};
typedef struct OTISDNAddress OTISDNAddress;
```

Fields

`fAddressType`

`fPhoneLength`

`fPhoneNumber`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProviders.h`

OTItemCount

```
typedef ItemCount OTItemCount;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

LIFO List Structure

Supports last-in, first-out lists in Open Transport.

```
struct OTLIFO {
    OTLink * fHead;
};
typedef struct OTLIFO OTLIFO;
```

Fields
`fHead`
A pointer to the first entry in the linked list. Set this to nil to initialize the structure before using it.

Discussion
Open Transport LIFO (last-in, first-out) lists use the LIFO list structure. You must initialize this structure by setting the structure's fHead field to NULL before using the LIFO list. Most Open Transport LIFO list operations are atomic.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

OTLink

Allows any data structure to be used in an Open Transport list.

```
struct OTLink {
    OTLink * fNext;
};
typedef struct OTLink OTLink;
```

Fields
`fNext`
A pointer to the next entry in the linked list.

Discussion
All of Open Transport's list utilities use the linked list structure, which may be embedded in any data structure that you want to use in an Open Transport list. A linked list structure is defined by the OTLink data type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

FIFO List Structure

Supports first-in, last-out lists in Open Transport

```
struct OTList {
    OTLink * fHead;
};
typedef struct OTList OTList;
```

Fields
fHead
A pointer to the first entry in the linked list. Set this to NULL to initialize the structure before using it.

Discussion
Open Transport FIFO (first-in, first-out) lists use the FIFO list structure. You must initialize this structure by setting the structure's fHead field to NULL before using the LIFO list. The FIFO list structure is defined by the OTList data type.

None of the functions that handle a FIFO list structure are atomic.

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTListSearchUPP

```
typedef OTListSearchProcPtr OTListSearchUPP;
```

Discussion
For more information, see the description of the OTListSearchUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

Lock Data Type

Defines a value used to ensure that Open Transport does not recursively reenter locked areas of code.

```
typedef UInt8 OTLock;
```

Discussion
The lock data type defines a value that is used by the OTClearLock function and the OTAcquireLock function to ensure that Open Transport does not recursively reenter locked areas of code. The lock data type is defined by the OTLock data type

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

OTNameID

```
typedef SInt32 OTNameID;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

OTNotifyUPP

```
typedef OTNotifyProcPtr OTNotifyUPP;
```

Discussion
For more information, see the description of the OTNotifyUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

OTPCIInfo

```
struct OTPCIInfo {
    RegEntryID fTheID;
    void *fConfigurationInfo;
    ByteCount fConfigurationLength;
};
```

Fields

OTPortCloseStruct

Denies or accepts requests to yield a port.

```
struct OTPortCloseStruct {
    OTPortRef fPortRef;
    ProviderRef fTheProvider;
    OSStatus fDenyReason;
};
typedef struct OTPortCloseStruct OTPortCloseStruct;
```

Fields

`fPortRef`

The port requested to be closed.

`fTheProvider`

The provider that is currently using the port.

`fDenyReason`

A value that you can leave untouched to accept the yield request. To deny the request, change this value to a negative error code corresponding to the reason for your denial (normally you use the kOTUserRequestedErr error).

Discussion

When you are using a port that another client wishes to use, the other client can use the OTYieldPortRequest function (not available in Mac OS X) to ask you to yield the port. If you are registered as a client of Open Transport, you receive a kOTYieldPortRequest event, whose cookie parameter is a pointer to a port close structure. You can use this structure to deny or accept the yield request.

Currently, this callback is only used for serial ports, but it is applicable to any hardware device that cannot share a port with multiple clients. You should check the kOTCanYieldPort bit in the port structure's fInfoFlags field to see whether the port supports yielding.

If the provider is passively listening (that is, bound with a queue length (qlen) greater than 0) and you are willing to yield, you need do nothing. If, however, you are actively connected and you are willing to yield the port, you must issue a synchronous OTSndDisconnect callin order to let the port go.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

The Port Structure

Describes a port's characteristics.

```
struct OTPortRecord {
    OTPortRef fRef;
    UInt32 fPortFlags;
    UInt32 fInfoFlags;
    UInt32 fCapabilities;
    ItemCount fNumChildPorts;
    OTPortRef * fChildPorts;
    char fPortName[36];
    char fModuleName[32];
    char fSlotID[8];
    char fResourceInfo[32];
    char fReserved[164];
};
typedef struct OTPortRecord OTPortRecord;
```

Fields

`fRef`

The port reference; a 32-bit value encoding the port's device type, bus type, slot number, and multiport identifier

`fPortFlags`

Flags describing the port's status. If no bits are set, the port is currently inactive—that is, it is not in use at this time.

`fInfoFlags`

`fCapabilities`

`fNumChildPorts`

`fChildPorts`

An array of the port references for the child ports associated with this port. When you get a Port Record, this pointer typically points into the SReserved field at the end of the record.

`fPortName`

A unique name for this port. The port name is a zero-terminated string that can have a maximum length as indicated by the constant kMaxProviderNameSize.

`fModuleName`

The name of the actual STREAMS module that implements the driver for this port. Open Transport uses this name internally; applications rarely need to use this name.

`fSlotID`

An 8-byte identifier for a port's slot that contains a 7-byte character string plus a zero for termination. This identifier is typically available only for PCI cards.

`fResourceInfo`

A zero-terminated string that describes a shared library that can handle configuration information for the device. This field contains an identifier that allows Open Transport to access auxiliary information about the driver (Open Transport creates shared library IDs from this string to be able to find these extra shared libraries). This string should either be unique to the driver or should be set to a NULL string.

`fReserved`

Reserved.

Discussion

Open Transport uses a port structure to describe a port's characteristics, such as its port name, its child ports, whether it is active or disabled, whether it is private or shareable, and the kind of framing it can use.

Availability

Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTPortRef

```
typedef UInt32 OTPortRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTProcessUPP

```
typedef OTProcessProcPtr OTProcessUPP;
```

Discussion
For more information, see the description of the OTProcessUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTQLen

```
typedef UInt32 OTQLen;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTReadInfo

```
struct OTReadInfo {
    UInt32 fType;
    OTCommand fCommand;
    UInt32 fFiller;
    ByteCount fBytes;
    OSStatus fError;
};
typedef struct OTReadInfo OTReadInfo;
```

Fields

`fType`

`fCommand`

`fFiller`

`fBytes`

`fError`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

OTReason

```
typedef SInt32 OTReason;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

OTResourceLocator

```
struct OTResourceLocator {
    FSSpec fFile;
    UInt16 fResID;
};
typedef struct OTResourceLocator OTResourceLocator;
```

Fields

`fFile`

`fResID`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

OTResult

```
typedef SInt32 OTResult;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTScriptInfo

```
struct OTScriptInfo {
    UInt32 fScriptType;
    void * fTheScript;
    UInt32 fScriptLength;
};
typedef struct OTScriptInfo OTScriptInfo;
```

Fields
fScriptType
fTheScript
fScriptLength

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTSequence

```
typedef SInt32 OTSequence;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTSInt16Param

```
typedef SInt16 OTSInt16Param;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTSInt8Param

```
typedef SInt8 OTSInt8Param;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTSlotNumber

```
typedef UInt16 OTSlotNumber;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTStateMachine

```
struct OTStateMachine {
    OTStateMachineDataPad fData;
    void * fCookie;
    OTEventCode fCode;
    OTResult fResult;
};
typedef struct OTStateMachine OTStateMachine;
```

Fields
fData
fCookie
fCode
fResult

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

OTStateMachineDataPad

```
typedef UInt8 OTStateMachineDataPad[12];
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

OTSystemTaskRef

```
typedef  OTSystemTaskRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

OTTimeout

```
typedef UInt32 OTTimeout;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

OTTimerTask

```
typedef  OTTimerTask;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

Timestamp Data Type

Contains an Open Transport timestamp.

```
typedef UnsignedWide OTTimeStamp;
```

Discussion
The timestamp data type is a 64-bit value that contains an Open Transport timestamp. The timestamp has unspecified units; you must use one of the timestamp manipulation functions described in "Timestamp Utility Functions" to convert the timestamp to known quantities. The timestamp data type is defined by the OTTimeStamp data type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

OTUInt16Param

```
typedef UInt16 OTUInt16Param;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTUInt32

```
typedef UInt32 OTUInt32;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTUInt8Param

```
typedef UInt8 OTUInt8Param;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTUnixErr

```
typedef UInt16 OTUnixErr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTXTILevel

```
typedef UInt32 OTXTILevel;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

OTXTIName

```
typedef UInt32 OTXTIName;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

pollfd

```
struct pollfd {
    SInt32 fd;
    SInt16 events;
    SInt16 revents;
    SInt32 _ifd;
};
```

Fields

PollRef

```
struct PollRef {
    SInt32 filler;
    SInt16 events;
    SInt16 revents;
    StreamRef ref;
};
typedef struct PollRef PollRef;
```

Fields
`filler`
`events`
`revents`
`ref`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

PPPMRULimits

```
struct PPPMRULimits {
    UInt32 mruSize;
    UInt32 upperMRULimit;
    UInt32 lowerMRULimit;
};
typedef struct PPPMRULimits PPPMRULimits;
```

Fields
`mruSize`
`upperMRULimit`
`lowerMRULimit`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProviders.h`

ProviderRef

```
typedef struct OpaqueProviderRef * ProviderRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

q_xtra

```
struct q_xtra {
    UInt32 dummy;
};
typedef struct q_xtra q_xtra;
```

Fields
`dummy`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

qband

```
struct qband {
    qband * qb_next;
    unsigned long qb_count;
    msgb * qb_first;
    msgb * qb_last;
    unsigned long qb_hiwat;
    unsigned long qb_lowat;
    unsigned short qb_flag;
    short qb_pad1;
};
typedef struct qband qband;
typedef qband qband_t;
```

Fields

`qb_next`

`qb_count`

`qb_first`

`qb_last`

`qb_hiwat`

`qb_lowat`

`qb_flag`

`qb_pad1`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

qband_t

```
typedef qband qband_t;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

qfields_t

```
typedef qfields qfields_t;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

qinit

```
struct qinit {
    putp_t qi_putp;
    srvp_t qi_srvp;
    openp_t qi_qopen;
    closep_t qi_qclose;
    admin_t qi_qadmin;
    module_info * qi_minfo;
    module_stat * qi_mstat;
};
typedef struct qinit qinit;
```

Fields

`qi_putp`

`qi_srvp`

`qi_qopen`

`qi_qclose`

`qi_qadmin`

`qi_minfo`

`qi_mstat`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

queue

```
struct queue {
    qinit * q_qinfo;
    msgb * q_first;
    msgb * q_last;
    queue * q_next;
    queue_q_u q_u;
    char * q_ptr;
    unsigned long q_count;
    long q_minpsz;
    long q_maxpsz;
    unsigned long q_hiwat;
    unsigned long q_lowat;
    qband * q_bandp;
    unsigned short q_flag;
    unsigned char q_nband;
    unsigned char q_pad1[1];
    q_xtra * q_osx;
    queue * q_ffcp;
    queue * q_bfcp;
};
typedef struct queue queue;
typedef queue * queuePtr;
```

Fields

`q_qinfo`

`q_first`

`q_last`

`q_next`

`q_u`

`q_ptr`

`q_count`

`q_minpsz`

`q_maxpsz`

`q_hiwat`

`q_lowat`

`q_bandp`

`q_flag`

`q_nband`

`q_pad1`

`q_osx`

`q_ffcp`

`q_bfcp`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

queue_q_u

```
union queue_q_u {
    queue * q_u_link;
    sqh_s * q_u_sqh_parent;
};
typedef union queue_q_u queue_q_u;
```

Fields
`q_u_link`
`q_u_sqh_parent`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

queue_t

```
typedef SInt32 queue_t;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`queue.h`

short_p

```
typedef SInt16 short_p;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

sqh_s

```
struct sqh_s {
    UInt32 dummy;
};
typedef struct sqh_s sqh_s;
```

Fields
`dummy`

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

sth_s

```
struct sth_s {
    UInt32 dummy;
};
typedef struct sth_s sth_s;
```

Fields
dummy

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

str_list

```
struct str_list {
    SInt32 sl_nmods;
    str_mlist * sl_modlist;
};
typedef struct str_list str_list;
```

Fields
sl_nmods
sl_modlist

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

str_mlist

```
struct str_mlist {
    char l_name[32];
};
typedef struct str_mlist str_mlist;
```

Fields
l_name

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

strbuf

```
struct strbuf {
    SInt32 maxlen;
    SInt32 len;
    char * buf;
};
typedef struct strbuf strbuf;
```

Fields

`maxlen`

`len`

`buf`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransport.h`

StreamRef

```
typedef struct OpaqueStreamRef * StreamRef;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

streamtab

```
struct streamtab {
    qinit * st_rdinit;
    qinit * st_wrinit;
    qinit * st_muxrinit;
    qinit * st_muxwinit;
};
typedef struct streamtab streamtab;
```

Fields

`st_rdinit`

`st_wrinit`

`st_muxrinit`

`st_muxwinit`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

strfdinsert

```
struct strfdinsert {
    strbuf ctlbuf;
    strbuf databuf;
    long flags;
    long fildes;
    SInt32 offset;
};
typedef struct strfdinsert strfdinsert;
```

Fields

`ctlbuf`

`databuf`

`flags`

`fildes`

`offset`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

strioctl

```
struct strioctl {
    SInt32 ic_cmd;
    SInt32 ic_timout;
    SInt32 ic_len;
    char * ic_dp;
};
typedef struct strioctl strioctl;
```

Fields

`ic_cmd`

`ic_timout`

`ic_len`

`ic_dp`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

stroptions

```
struct stroptions {
    unsigned long so_flags;
    short so_readopt;
    unsigned short so_wroff;
    long so_minpsz;
    long so_maxpsz;
    unsigned long so_hiwat;
    unsigned long so_lowat;
    unsigned char so_band;
    unsigned char so_filler[3];
    unsigned long so_poll_set;
    unsigned long so_poll_clr;
};
typedef struct stroptions stroptions;
```

Fields

`so_flags`

`so_readopt`

`so_wroff`

`so_minpsz`

`so_maxpsz`

`so_hiwat`

`so_lowat`

`so_band`

`so_filler`

`so_poll_set`

`so_poll_clr`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

strpeek

```
struct strpeek {
    strbuf ctlbuf;
    strbuf databuf;
    long flags;
};
typedef struct strpeek strpeek;
```

Fields

`ctlbuf`

`databuf`

`flags`

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

strpfp

```
struct strpfp {
    unsigned long pass_file_cookie;
    unsigned short pass_uid;
    unsigned short pass_gid;
    sth_s * pass_sth;
};
typedef struct strpfp strpfp;
```

Fields
pass_file_cookie
pass_uid
pass_gid
pass_sth

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

strpmsg

```
struct strpmsg {
    strbuf ctlbuf;
    strbuf databuf;
    SInt32 band;
    long flags;
};
typedef struct strpmsg strpmsg;
```

Fields
ctlbuf
databuf
band
flags

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

strrecvfd

```
struct strrecvfd {
    long fd;
    unsigned short uid;
    unsigned short gid;
    char fill[8];
};
typedef struct strrecvfd strrecvfd;
```

Fields

`fd`

`uid`

`gid`

`fill`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T_addr_ack

```
struct T_addr_ack {
    long PRIM_type;
    long LOCADDR_length;
    long LOCADDR_offset;
    long REMADDR_length;
    long REMADDR_offset;
};
typedef struct T_addr_ack T_addr_ack;
```

Fields

`PRIM_type`

`LOCADDR_length`

`LOCADDR_offset`

`REMADDR_length`

`REMADDR_offset`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T_addr_req

```
struct T_addr_req {
    long PRIM_type;
};
typedef struct T_addr_req T_addr_req;
```

Fields
`PRIM_type`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

T_bind_ack

```
struct T_bind_ack {
    long PRIM_type;
    long ADDR_length;
    long ADDR_offset;
    unsigned long CONIND_number;
};
typedef struct T_bind_ack T_bind_ack;
```

Fields
`PRIM_type`
`ADDR_length`
`ADDR_offset`
`CONIND_number`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

T_bind_req

```
struct T_bind_req {
    long PRIM_type;
    long ADDR_length;
    long ADDR_offset;
    unsigned long CONIND_number;
};
typedef struct T_bind_req T_bind_req;
```

Fields

`PRIM_type`

`ADDR_length`

`ADDR_offset`

`CONIND_number`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

t_call

Fields

T_cancelreply_req

```
struct T_cancelreply_req {
    long PRIM_type;
    long SEQ_number;
};
typedef struct T_cancelreply_req T_cancelreply_req;
```

Fields

`PRIM_type`

`SEQ_number`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T_cancelrequest_req

```
struct T_cancelrequest_req {
    long PRIM_type;
    long SEQ_number;
};
typedef struct T_cancelrequest_req T_cancelrequest_req;
```

Fields

`PRIM_type`

`SEQ_number`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T_conn_con

```
struct T_conn_con {
    long PRIM_type;
    long RES_length;
    long RES_offset;
    long OPT_length;
    long OPT_offset;
};
typedef struct T_conn_con T_conn_con;
```

Fields

`PRIM_type`

`RES_length`

`RES_offset`

`OPT_length`

`OPT_offset`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T_conn_ind

```
struct T_conn_ind {
    long PRIM_type;
    long SRC_length;
    long SRC_offset;
    long OPT_length;
    long OPT_offset;
    long SEQ_number;
};
typedef struct T_conn_ind T_conn_ind;
```

Fields

`PRIM_type`

`SRC_length`

`SRC_offset`

`OPT_length`

`OPT_offset`

`SEQ_number`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T_conn_req

```
struct T_conn_req {
    long PRIM_type;
    long DEST_length;
    long DEST_offset;
    long OPT_length;
    long OPT_offset;
};
typedef struct T_conn_req T_conn_req;
```

Fields

`PRIM_type`

`DEST_length`

`DEST_offset`

`OPT_length`

`OPT_offset`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T_conn_res

```
struct T_conn_res {
    long PRIM_type;
    queue_t * QUEUE_ptr;
    long OPT_length;
    long OPT_offset;
    long SEQ_number;
};
typedef struct T_conn_res T_conn_res;
```

Fields

`PRIM_type`

`QUEUE_ptr`

`OPT_length`

`OPT_offset`

`SEQ_number`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T_data_ind

```
struct T_data_ind {
    long PRIM_type;
    long MORE_flag;
};
typedef struct T_data_ind T_data_ind;
```

Fields

`PRIM_type`

`MORE_flag`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T_data_req

```
struct T_data_req {
    long PRIM_type;
    long MORE_flag;
};
typedef struct T_data_req T_data_req;
```

Fields

`PRIM_type`

`MORE_flag`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T_delname_req

```
struct T_delname_req {
    long PRIM_type;
    long SEQ_number;
    long NAME_length;
    long NAME_offset;
};
typedef struct T_delname_req T_delname_req;
```

Fields

`PRIM_type`

`SEQ_number`

`NAME_length`

`NAME_offset`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

t_discon

Fields

T_discon_ind

```
struct T_discon_ind {
    long PRIM_type;
    long DISCON_reason;
    long SEQ_number;
};
typedef struct T_discon_ind T_discon_ind;
```

Fields

`PRIM_type`

`DISCON_reason`

`SEQ_number`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

T_discon_req

```
struct T_discon_req {
    long PRIM_type;
    long SEQ_number;
};
typedef struct T_discon_req T_discon_req;
```

Fields

`PRIM_type`

`SEQ_number`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

T_error_ack

```
struct T_error_ack {
    long PRIM_type;
    long ERROR_prim;
    long TLI_error;
    long UNIX_error;
};
typedef struct T_error_ack T_error_ack;
```

Fields

`PRIM_type`

`ERROR_prim`

`TLI_error`

`UNIX_error`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T_event_ind

```
struct T_event_ind {
    long PRIM_type;
    long EVENT_code;
    long EVENT_cookie;
};
typedef struct T_event_ind T_event_ind;
```

Fields

`PRIM_type`

`EVENT_code`

`EVENT_cookie`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T_exdata_ind

```
struct T_exdata_ind {
    long PRIM_type;
    long MORE_flag;
};
typedef struct T_exdata_ind T_exdata_ind;
```

Fields

`PRIM_type`

`MORE_flag`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T_exdata_req

```
struct T_exdata_req {
    long PRIM_type;
    long MORE_flag;
};
typedef struct T_exdata_req T_exdata_req;
```

Fields

`PRIM_type`

`MORE_flag`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

t_info

Fields

T_info_ack

```
struct T_info_ack {
    long PRIM_type;
    long TSDU_size;
    long ETSDU_size;
    long CDATA_size;
    long DDATA_size;
    long ADDR_size;
    long OPT_size;
    long TIDU_size;
    long SERV_type;
    long CURRENT_state;
    long PROVIDER_flag;
};
typedef struct T_info_ack T_info_ack;
```

Fields

PRIM_type

TSDU_size

ETSDU_size

CDATA_size

DDATA_size

ADDR_size

OPT_size

TIDU_size

SERV_type

CURRENT_state

PROVIDER_flag

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

T_info_req

```
struct T_info_req {
    long PRIM_type;
};
typedef struct T_info_req T_info_req;
```

Fields

PRIM_type

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

The Keepalive Structure

Specifies the value of the `OPT_KEEPALIVE` option.

```
struct t_kpalive {
    SInt32 kp_onoff;
    SInt32 kp_timeout;
};
typedef struct t_kpalive t_kpalive;
```

Fields

`kp_onoff`

A constant specifying whether the option is turned on (T_ON) or off (T_OFF).

`kp_timeout`

A positive integer specifying how many minutes Open Transport should maintain a connection in the absence of traffic.

Discussion

The keepalive structure specifies the value of the OPT_KEEPALIVE option, described in "XTI-Level Options and Generic Options" (page 2718)

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransport.h`

The Linger Structure

Specifies the value of the `XTI_LINGER` option.

```
struct t_linger {
    SInt32 l_onoff;
    SInt32 l_linger;
};
typedef struct t_linger t_linger;
```

Fields

`l_onoff`

A constant specifying whether the option is turned on (T_ON) or off (T_OFF).

`l_linger`

An integer specifying the linger time, the amount of time in seconds that Open Transport should wait to allow data in an endpoint's internal buffer to be sent before the OTCloseProvider function closes the endpoint.

To request the default value for this option, set the l_linger field to T_UNSPEC.

Discussion

The linger structure specifies the value of the `XTI_LINGER` option, described in "XTI-Level Options and Generic Options" (page 2718).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransport.h`

T_lkupname_con

```
struct T_lkupname_con {
    long PRIM_type;
    long SEQ_number;
    long NAME_length;
    long NAME_offset;
    long RSP_count;
    long RSP_cumcount;
};
typedef struct T_lkupname_con T_lkupname_con;
```

Fields

`PRIM_type`

`SEQ_number`

`NAME_length`

`NAME_offset`

`RSP_count`

`RSP_cumcount`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T_lkupname_req

```
struct T_lkupname_req {
    long PRIM_type;
    long SEQ_number;
    long NAME_length;
    long NAME_offset;
    long ADDR_length;
    long ADDR_offset;
    long MAX_number;
    long MAX_milliseconds;
    long REQ_flags;
};
typedef struct T_lkupname_req T_lkupname_req;
```

Fields

`PRIM_type`

`SEQ_number`

`NAME_length`

`NAME_offset`

`ADDR_length`

`ADDR_offset`

`MAX_number`

`MAX_milliseconds`

`REQ_flags`

Availability

Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

T_MIB_ack

Fields

T_MIB_req

Fields

T_ok_ack

```
struct T_ok_ack {
    long PRIM_type;
    long CORRECT_prim;
};
typedef struct T_ok_ack T_ok_ack;
```

Fields
PRIM_type
CORRECT_prim

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

t_opthdr

Fields

T_optmgmt_ack

```
struct T_optmgmt_ack {
    long PRIM_type;
    long OPT_length;
    long OPT_offset;
    long MGMT_flags;
};
typedef struct T_optmgmt_ack T_optmgmt_ack;
```

Fields

`PRIM_type`

`OPT_length`

`OPT_offset`

`MGMT_flags`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

T_optmgmt_req

```
struct T_optmgmt_req {
    long PRIM_type;
    long OPT_length;
    long OPT_offset;
    long MGMT_flags;
};
typedef struct T_optmgmt_req T_optmgmt_req;
```

Fields

`PRIM_type`

`OPT_length`

`OPT_offset`

`MGMT_flags`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

T_ordrel_ind

```
struct T_ordrel_ind {
    long PRIM_type;
};
typedef struct T_ordrel_ind T_ordrel_ind;
```

Fields
`PRIM_type`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

T_ordrel_req

```
struct T_ordrel_req {
    long PRIM_type;
};
typedef struct T_ordrel_req T_ordrel_req;
```

Fields
`PRIM_type`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

T_primitives

```
union T_primitives {
    long primType;
    T_addr_ack taddrack;
    T_bind_ack tbindack;
    T_bind_req tbindreq;
    T_conn_con tconncon;
    T_conn_ind tconnind;
    T_conn_req tconnreq;
    T_conn_res tconnres;
    T_data_ind tdataind;
    T_data_req tdatareq;
    T_discon_ind tdisconind;
    T_discon_req tdisconreq;
    T_exdata_ind texdataind;
    T_exdata_req texdatareq;
    T_error_ack terrorack;
    T_info_ack tinfoack;
    T_info_req tinforeq;
    T_ok_ack tokack;
    T_optmgmt_ack toptmgmtack;
    T_optmgmt_req toptmgmtreq;
    T_ordrel_ind tordrelind;
    T_ordrel_req tordrelreq;
    T_unbind_req tunbindreq;
    T_uderror_ind tuderrorind;
    T_unitdata_ind tunitdataind;
    T_unitdata_req tunitdatareq;
    T_unitreply_ind tunitreplyind;
    T_unitrequest_ind tunitrequestind;
    T_unitrequest_req tunitrequestreq;
    T_unitreply_req tunitreplyreq;
    T_unitreply_ack tunitreplyack;
    T_reply_ind treplyind;
    T_request_ind trequestind;
    T_request_req trequestreq;
    T_reply_req treplyreq;
    T_reply_ack treplyack;
    T_cancelrequest_req tcancelreqreq;
    T_resolveaddr_req tresolvereq;
    T_resolveaddr_ack tresolveack;
    T_regname_req tregnamereq;
    T_regname_ack tregnameack;
    T_delname_req tdelnamereq;
    T_lkupname_req tlkupnamereq;
    T_lkupname_con tlkupnamecon;
    T_sequence_ack tsequenceack;
    T_event_ind teventind;
};
typedef union T_primitives T_primitives;
```

Fields

`primType`

`taddrack`

`tbindack`

`tbindreq`

```
tconncon
tconnind
tconnreq
tconnres
tdataind
tdatareq
tdisconind
tdisconreq
texdataind
texdatareq
terrorack
tinfoack
tinforeq
tokack
toptmgmtack
toptmgmtreq
tordrelind
tordrelreq
tunbindreq
tuderrorind
tunitdataind
tunitdatareq
tunitreplyind
tunitrequestind
tunitrequestreq
tunitreplyreq
tunitreplyack
treplyind
trequestind
trequestreq
treplyreq
treplyack
tcancelreqreq
tresolvereq
tresolveack
tregnamereq
tregnameack
tdelnamereq
tlkupnamereq
tlkupnamecon
tsequenceack
teventind
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

T_regname_ack

```
struct T_regname_ack {
    long PRIM_type;
    long SEQ_number;
    long REG_id;
    long ADDR_length;
    long ADDR_offset;
};
typedef struct T_regname_ack T_regname_ack;
```

Fields

`PRIM_type`

`SEQ_number`

`REG_id`

`ADDR_length`

`ADDR_offset`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T_regname_req

```
struct T_regname_req {
    long PRIM_type;
    long SEQ_number;
    long NAME_length;
    long NAME_offset;
    long ADDR_length;
    long ADDR_offset;
    long REQ_flags;
};
typedef struct T_regname_req T_regname_req;
```

Fields

`PRIM_type`

`SEQ_number`

`NAME_length`

`NAME_offset`

`ADDR_length`

`ADDR_offset`

`REQ_flags`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

t_reply

Fields

T_reply_ack

```
struct T_reply_ack {
    long PRIM_type;
    long SEQ_number;
    long TLI_error;
    long UNIX_error;
};
typedef struct T_reply_ack T_reply_ack;
```

Fields
PRIM_type
SEQ_number
TLI_error
UNIX_error

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

T_reply_ind

```
struct T_reply_ind {
    long PRIM_type;
    long SEQ_number;
    long OPT_length;
    long OPT_offset;
    long REP_flags;
    long TLI_error;
    long UNIX_error;
};
typedef struct T_reply_ind T_reply_ind;
```

Fields
PRIM_type
SEQ_number
OPT_length
OPT_offset
REP_flags
TLI_error
UNIX_error

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

T_reply_req

```
struct T_reply_req {
    long PRIM_type;
    long SEQ_number;
    long OPT_length;
    long OPT_offset;
    long REP_flags;
};
typedef struct T_reply_req T_reply_req;
```

Fields
PRIM_type
SEQ_number
OPT_length
OPT_offset
REP_flags

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

t_request

Fields

T_request_ind

```
struct T_request_ind {
    long PRIM_type;
    long SEQ_number;
    long OPT_length;
    long OPT_offset;
    long REQ_flags;
};
typedef struct T_request_ind T_request_ind;
```

Fields

`PRIM_type`

`SEQ_number`

`OPT_length`

`OPT_offset`

`REQ_flags`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

T_request_req

```
struct T_request_req {
    long PRIM_type;
    long SEQ_number;
    long OPT_length;
    long OPT_offset;
    long REQ_flags;
};
typedef struct T_request_req T_request_req;
```

Fields

`PRIM_type`

`SEQ_number`

`OPT_length`

`OPT_offset`

`REQ_flags`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

T_resolveaddr_ack

```
struct T_resolveaddr_ack {
    long PRIM_type;
    long SEQ_number;
    long ADDR_length;
    long ADDR_offset;
    long ORIG_client;
    long ORIG_data;
    long TLI_error;
    long UNIX_error;
};
typedef struct T_resolveaddr_ack T_resolveaddr_ack;
```

Fields

`PRIM_type`

`SEQ_number`

`ADDR_length`

`ADDR_offset`

`ORIG_client`

`ORIG_data`

`TLI_error`

`UNIX_error`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T_resolveaddr_req

```
struct T_resolveaddr_req {
    long PRIM_type;
    long SEQ_number;
    long ADDR_length;
    long ADDR_offset;
    long ORIG_client;
    long ORIG_data;
    long MAX_milliseconds;
};
typedef struct T_resolveaddr_req T_resolveaddr_req;
```

Fields

`PRIM_type`

`SEQ_number`

`ADDR_length`

`ADDR_offset`

`ORIG_client`

`ORIG_data`

`MAX_milliseconds`

Availability

Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

T_sequence_ack

```
struct T_sequence_ack {
    long PRIM_type;
    long ORIG_prim;
    long SEQ_number;
    long TLI_error;
    long UNIX_error;
};
typedef struct T_sequence_ack T_sequence_ack;
```

Fields
PRIM_type
ORIG_prim
SEQ_number
TLI_error
UNIX_error

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

T_stream_timer

Fields

T_stream_timer_1

Fields

t_uderr

Fields

T_uderror_ind

```
struct T_uderror_ind {
    long PRIM_type;
    long DEST_length;
    long DEST_offset;
    long OPT_length;
    long OPT_offset;
    long ERROR_type;
};
typedef struct T_uderror_ind T_uderror_ind;
```

Fields

`PRIM_type`

`DEST_length`

`DEST_offset`

`OPT_length`

`OPT_offset`

`ERROR_type`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

T_unbind_req

```
struct T_unbind_req {
    long PRIM_type;
};
typedef struct T_unbind_req T_unbind_req;
```

Fields

`PRIM_type`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

t_unitdata

Fields

T_unitdata_ind

```
struct T_unitdata_ind {
    long PRIM_type;
    long SRC_length;
    long SRC_offset;
    long OPT_length;
    long OPT_offset;
};
typedef struct T_unitdata_ind T_unitdata_ind;
```

Fields

`PRIM_type`

`SRC_length`

`SRC_offset`

`OPT_length`

`OPT_offset`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

T_unitdata_req

```
struct T_unitdata_req {
    long PRIM_type;
    long DEST_length;
    long DEST_offset;
    long OPT_length;
    long OPT_offset;
};
typedef struct T_unitdata_req T_unitdata_req;
```

Fields

`PRIM_type`

`DEST_length`

`DEST_offset`

`OPT_length`

`OPT_offset`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

t_unitreply

Fields

T_unitreply_ack

```
struct T_unitreply_ack {
    long PRIM_type;
    long SEQ_number;
    long TLI_error;
    long UNIX_error;
};
typedef struct T_unitreply_ack T_unitreply_ack;
```

Fields

`PRIM_type`

`SEQ_number`

`TLI_error`

`UNIX_error`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

T_unitreply_ind

```
struct T_unitreply_ind {
    long PRIM_type;
    long SEQ_number;
    long OPT_length;
    long OPT_offset;
    long REP_flags;
    long TLI_error;
    long UNIX_error;
};
typedef struct T_unitreply_ind T_unitreply_ind;
```

Fields

`PRIM_type`

`SEQ_number`

`OPT_length`

`OPT_offset`

`REP_flags`

`TLI_error`

`UNIX_error`

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

T_unitreply_req

```
struct T_unitreply_req {
    long PRIM_type;
    long SEQ_number;
    long OPT_length;
    long OPT_offset;
    long REP_flags;
};
typedef struct T_unitreply_req T_unitreply_req;
```

Fields
PRIM_type
SEQ_number
OPT_length
OPT_offset
REP_flags

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransportProtocol.h

t_unitrequest

Fields

T_unitrequest_ind

```
struct T_unitrequest_ind {
    long PRIM_type;
    long SEQ_number;
    long SRC_length;
    long SRC_offset;
    long OPT_length;
    long OPT_offset;
    long REQ_flags;
};
typedef struct T_unitrequest_ind T_unitrequest_ind;
```

Fields

`PRIM_type`

`SEQ_number`

`SRC_length`

`SRC_offset`

`OPT_length`

`OPT_offset`

`REQ_flags`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T_unitrequest_req

```
struct T_unitrequest_req {
    long PRIM_type;
    long SEQ_number;
    long DEST_length;
    long DEST_offset;
    long OPT_length;
    long OPT_offset;
    long REQ_flags;
};
typedef struct T_unitrequest_req T_unitrequest_req;
```

Fields

`PRIM_type`

`SEQ_number`

`DEST_length`

`DEST_offset`

`OPT_length`

`OPT_offset`

`REQ_flags`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProtocol.h`

T8022Address

```
struct T8022Address {
    OTAddressType fAddrFamily;
    UInt8 fHWAddr[6];
    UInt16 fSAP;
    UInt8 fSNAP[5];
};
typedef struct T8022Address T8022Address;
```

Fields

`fAddrFamily`

`fHWAddr`

`fSAP`

`fSNAP`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProviders.h`

T8022FullPacketHeader

```
struct T8022FullPacketHeader {
    EnetPacketHeader fEnetPart;
    T8022SNAPHeader f8022Part;
};
typedef struct T8022FullPacketHeader T8022FullPacketHeader;
```

Fields

`fEnetPart`

`f8022Part`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProviders.h`

T8022Header

```
struct T8022Header {
    UInt8 fDSAP;
    UInt8 fSSAP;
    UInt8 fCtrl;
};
typedef struct T8022Header T8022Header;
```

Fields

`fDSAP`

`fSSAP`

`fCtrl`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransportProviders.h`

T8022SNAPHeader

```
struct T8022SNAPHeader {
    UInt8 fDSAP;
    UInt8 fSSAP;
    UInt8 fCtrl;
    UInt8 fSNAP[5];
};
typedef struct T8022SNAPHeader T8022SNAPHeader;
```

Fields

`fDSAP`

`fSSAP`

`fCtrl`

`fSNAP`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProviders.h`

TBind

Describes the protocol address to which an endpoint is currently bound or connected, or specifies the protocol address to which you wish to bind or connect the endpoint.

```
struct TBind {
    TNetbuf addr;
    OTQLen qlen;
};
typedef struct TBind TBind;
```

Fields

`addr`

A `TNetbuf` structure that contains information about an address. The `addr.maxlen` field specifies the maximum size of the address, the `addr.len` field specifies the actual length of the address, and the `addr.buf` field points to the buffer containing the address.

When specifying an address, you must allocate a buffer for the address and initialize it; you must set the `addr.buf` field to point to this buffer; and you must set the `addr.len` field to the size of the address.

When requesting an address, you must allocate a buffer in which the address is to be placed; you must set the `addr.buf` field to point to this buffer; and you must set the `addr.maxlen` field to the maximum size of the address that is being returned. You determine this value by examining the `addr` field of the `TEndpointInfo` (page 2542) structure for the endpoint.

`qlen`

For a connection-oriented endpoint, the maximum number of connection requests that can be concurrently outstanding for this endpoint. For more information, see the description of the `OTBind` (page 2319) function. For connectionless endpoints, this field has no meaning.

Discussion
The `TBind` structure describes the protocol address to which an endpoint is currently bound or connected, or specifies the protocol address to which you wish to bind or connect the endpoint. For a connection-oriented endpoint, the `TBind` structure also specifies the actual or desired number of connection requests that can be concurrently outstanding for the endpoint.

You pass the `TBind` structure as a parameter to the `OTBind` (page 2319) function, the `OTGetProtAddress` (page 2348) function, and the `OTResolveAddress` (page 2387) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

TCall

Specifies the options and data associated with establishing a connection.

```
struct TCall {
    TNetbuf addr;
    TNetbuf opt;
    TNetbuf udata;
    OTSequence sequence;
};
typedef struct TCall TCall;
```

Fields

`addr`
A TNetbuf structure that specifies the location and size of an address buffer

`opt`
A TNetbuf structure that specifies the location and size of an options buffer.

`udata`
A TNetbuf structure that specifies the location and size of a buffer for data associated with a connection or disconnection request.

`sequence`
A 32-bit value used by the OTListen and OTAccept functions to specify the connection ID.

Discussion
You use the `TCall` structure to specify the options and data associated with establishing a connection. You pass a pointer to this structure as a parameter to the OTConnect function, the OTRcvConnect function, the OTListen function, and the OTAccept function.

If you are using the TCall structure to send information, you must allocate a buffer and initialize it to contain the information. Set the .buf field of each TNetbuf to point to the buffer, and then specify the size of the buffer using the .len field. Set this field to 0 if you are not sending data.

If you are using the TCall structure to receive information, you must allocate a buffer into which the function can place the information when it returns. Then set the .buf field of all the TNetbufs to point to this buffer, and set the .maxlen field to the maximum size of the information. Set this field to 0 if you are not interested in receiving information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

TDiscon

Specifies data sent with the OTSndDisconnect function and retrieved by the OTRcvDisconnect function.

```
struct TDiscon {
    TNetbuf udata;
    OTReason reason;
    OTSequence sequence;
};
typedef struct TDiscon TDiscon;
```

Fields

`udata`
A TNetbuf structure that references data sent with the OTSndDisconnect function or received by the OTRcvDisconnect function.

`reason`
A 32-bit value specifying an error code that identifies the reason for the disconnection. These codes are supplied by the protocol. For additional information, consult the documentation provided for the protocol you are using.

`sequence`
A 32-bit value specifying an outstanding connection request that has been rejected. This field is meaningful only when you have issued several connection requests to the same endpoint and are awaiting the results.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

TEndpointInfo

Describes the initial characteristics of an endpoint that you opened by calling the `OTOpenEndpointInContext` function; returned by calling `OTGetEndpointInfo`.

```
struct TEndpointInfo {
    OTDataSize addr;
    OTDataSize options;
    OTDataSize tsdu;
    OTDataSize etsdu;
    OTDataSize connect;
    OTDataSize discon;
    OTServiceType servtype;
    UInt32 flags;
};
typedef struct TEndpointInfo TEndpointInfo;
```

Fields

`addr`

`options`

A value greater than or equal to 0 indicates the maximum number of bytes needed to store the protocol-specific options that this endpoint supports, if any. A value of `T_INVALID` (-2) indicates that this endpoint has no protocol-specific options that you can set; they are read-only. A value of -3 specifies that the provider does not support any options.

`tsdu`

For a transactionless endpoint, a positive value indicates the maximum number of bytes in a transport service data unit (TSDU) for this endpoint. A value of `T_INFINITE` (-1) indicates that there is no limit to the size of a TSDU. A value of 0 indicates that the provider does not support the concept of a TSDU. This means that you cannot send data with logical boundaries preserved across a connection. A value of `T_INVALID` indicates that this endpoint cannot transfer normal data (as opposed to expedited data).

For a transaction-based endpoint, this field indicates the maximum number of bytes in a response.

`etsdu`

For a transactionless endpoint, a positive value indicates the maximum number of bytes in an expedited transport service data unit (ETSDU) for this endpoint. A value of `T_INFINITE` indicates that there is no limit to the size of a ETSDU. A value of 0 indicates that this endpoint does not support the concept of an ETSDU. This means that you must not send expedited data with logical boundaries preserved across a connection. A value of `T_INVALID` indicates that this endpoint cannot transfer expedited data.

For a transaction-based endpoint, this field indicates the maximum number of bytes in a request.

`connect`

For a connection-oriented endpoint, a value greater than or equal to 0 indicates the maximum amount of data (in bytes) that you can send with the `OTConnect` (page 2326) function or the `OTAccept` (page 2306) function. A value of `T_INVALID` indicates that this endpoint does not let you send data with these functions. This field is meaningless for other types of endpoints.

`discon`

For a connection-oriented endpoint, a value greater than or equal to 0 indicates the maximum amount of data (in bytes) that you can send using the `OTSndDisconnect` (page 2399) function. A value of `T_INVALID` indicates that this endpoint does not let you send data with disconnection requests. This field is meaningless for other types of endpoints.

`servtype`

A constant that indicates what kind of service the endpoint provides. Possible values are given by the "Endpoint Service Types" (page 2670) enumeration.

`flags`

A bit FIeld that provides additional information about the endpoint. Possible values are given by the "Endpoint Flags" (page 2706) enumeration.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

IP Multicast Address Structure

Supports adding and dropping membership in an IP multicast address.

```
struct TIPAddMulticast {
    InetHost multicastGroupAddress;
    InetHost interfaceAddress;
};
typedef struct TIPAddMulticast TIPAddMulticast;
```

Fields

`multicastGroupAddress`
The IP address of the multicast group for which you want to add or drop membership.

`interfaceAddress`
The IP address of the network interface that you are using for the multicast group.

Discussion
You use the IP multicast address structure with the IP_ADD_MEMBERSHIP and IP_DROP_MEMBERSHIP options when you are adding or dropping membership in an IP multicast address.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProviders.h`

TLookupBuffer

Defines the format of entries in the buffer passed back in the reply parameter of the `OTLookupName` function.

```
struct TLookupBuffer {
    UInt16 fAddressLength;
    UInt16 fNameLength;
    UInt8 fAddressBuffer[1];
};
typedef struct TLookupBuffer TLookupBuffer;
```

Fields

`fAddressLength`
Specifies the size of the address specified by the fAddressBuffer field.

`fNameLength`
Specifies the size of the name that is stored in the buffer following the fAddressBuffer field.

`fAddressBuffer`
The first byte of the address to which the entity whose name follows (in the buffer) is bound.

Discussion
The TLookupBuffer structure defines the format of entries in the buffer passed back in the reply parameter of the OTLookupName function. When you parse the buffer in which the OTLookupName function places the names it has found, you can cast it as a TLookupBuffer structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

TLookupReply

Stores information passed back to your application by the OTLookupName function.

```
struct TLookupReply {
    TNetbuf names;
    UInt32 rspcount;
};
typedef struct TLookupReply TLookupReply;
```

Fields

`names`
: A TNetbuf structure that specifies the size and location of a buffer into which the OTLookupName function, on return, places the names it has found. You must allocate the buffer into which the replies are stored when the function returns; you must set the names.buf field to point to it; and you must set the names.maxlen field to the size of the buffer.

`rspcount`
: A long specifying, on return, the number of names found.

Discussion
You use the TLookupReply structure to store information passed back to you by the OTLookupName function. The information includes both a pointer to a buffer (containing registered entity names matching the criterion specified with the TLookupRequest structure) and the number of names found.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

TLookupRequest

Specifies the entity name to be looked up by the `OTLookupName` function.

```
struct TLookupRequest {
    TNetbuf name;
    TNetbuf addr;
    UInt32 maxcnt;
    OTTimeout timeout;
    OTFlags flags;
};
typedef struct TLookupRequest TLookupRequest;
```

Fields

`name`

A TNetbuf structure specifying the location and size of a buffer that contains the name to be looked up. You must allocate a buffer that contains the name, set the name.buf field to point to it, and set the name.len field to the length of the name.

`addr`

A TNetbuf structure describing the address of the node where you expect the names to be stored. You should normally supply 0 for addr.len. This causes the provider to use internal defaults for the starting point of the search. For a protocol family such as AppleTalk, in which every node has access to name and address information, this parameter is meaningless.

Specifying an address has meaning for those protocols that use a dedicated server or other device to store name information. In such a case, the name specified would override the protocol's default address. To specify an address, you would need to allocate a buffer containing the address, set the addr.buf field to point to it, and set the addr.len field to the length of the address. Consult the documentation supplied with your protocol to determine whether you can or should specify an address.

`maxcnt`

A long specifying the number of names you expect to be returned. Some protocols allow the use of wildcard characters in specifying a name. As a result, the OTLookupName function might find multiple names matching the specified name pattern. If you expect a specific number of replies for a particular name or do not expect to exceed a specific number, you should specify this number to obtain faster execution. Otherwise, set this field to 0xffff ffff; in this case, the timeout value will control the lookup.

`timeout`

A long specifying the amount of time, in milliseconds, that should elapse before Open Transport gives up searching for a name. Specify 0 to leave the timeout value up to the underlying naming system.

`flags`

Discussion

You use the TLookupRequest structure to specify the entity name to be looked up by the OTLookupName function and to set additional values that the mapper provider uses to circumscribe the search.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransport.h`

TNetbuf

Specifies the location and size of a buffer that contains an address, option information, or user data.

```
struct TNetbuf {
    ByteCount maxlen;
    ByteCount len;
    UInt8 * buf;
};
typedef struct TNetbuf TNetbuf;
```

Fields

`maxlen`

The size (in bytes) of the buffer to which the `buf` field points. You must set the `maxlen` field before passing a `TNetbuf` structure to a provider function as an output parameter. Open Transport ignores this field if you pass the `TNetbuf` structure as an input parameter.

`len`

The actual length (in bytes) of the information in the buffer to which the `buf` field points. If you are using the `TNetbuf` structure as an input parameter, you must set this field.

If you pass the `TNetbuf` structure as an output parameter, on return the provider function sets this field to the number of bytes the function has actually placed in the buffer referenced by the `buf` field.

`buf`

A pointer to a buffer. You must make sure that the `buf` field points to a valid buffer and that the buffer is large enough to store the information for which it is intended.

Discussion

You use a TNetbuf structure to specify the location and size of a buffer that contains an address, option information, or user data. Provider functions use TNetbuf structures both as input parameters and output parameters. If you use a TNetbuf structure as an input parameter, you specify the location and size of a buffer containing information you want to send. If you use a TNetbuf structure as an output parameter, you specify the location and the maximum size of the buffer used to hold information when the function returns.

You use a TNetbuf structure to describe the location and size of contiguous data. Open Transport allows you to describe noncontiguous data with the OTData structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransport.h`

The TOption Structure

Stores information about a single option in a buffer.

```
struct TOption {
    ByteCount len;
    OTXTILevel level;
    OTXTIName name;
    UInt32 status;
    UInt32 value[1];
};
typedef struct TOption TOption;
```

Fields

`len`

The size (in bytes) of the option information, including the header.

`level`
The protocol for which the option is defined.

`name`
The name of the option.

`status`
A status code specifying whether the negotiation has succeeded or failed. Possible values are given by the "Open Transport Flags and Status Codes" (page 2702) enumeration

`value`
The option value. To have the endpoint select an appropriate value, you can specify the constant T_UNSPEC.

Discussion
The TOption structure stores information about a single option in a buffer. All functions that you use to change or verify option values use a buffer containing TOption structures to store option information. For each option in the buffer, the TOption structure specifies the total length occupied by the option, the protocol level of the option, the option name, the status of a negotiated value, and the value of the option.

You use the TOption structure with the OPT_NEXTHDR macro, the OTCreateOptionString function, the OTNextOption function, and the OTFindOption function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

The TOptionHeader Structure

Stores information about options in a buffer.

```
struct TOptionHeader {
    ByteCount len;
    OTXTILevel level;
    OTXTIName name;
    UInt32 status;
};
typedef struct TOptionHeader TOptionHeader;
```

Fields
`len`
The size (in bytes) of the option information, including the header.

`level`
The protocol affected.

`name`
The option name.

`status`
The status value. Possible values are given by the "Open Transport Flags and Status Codes" (page 2702).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

The Option Management Structure

Manages the `req` and `ret` parameters of the `OTOptionManagement` function

```
struct TOptMgmt {
    TNetbuf opt;
    OTFlags flags;
};
typedef struct TOptMgmt TOptMgmt;
```

Fields

`opt`

A TNetbuf structure describing the buffer containing option information. The opt.maxlen field specifies the maximum size of the buffer. The opt.len field specifies the actual size of the buffer, and the opt.buf field contains the address of the buffer.

On input, as part of the req parameter, the buffer contains TOption structures describing the options to be negotiated or verified, or contains the names of options whose default or current values you are interested in. You must allocate this buffer, place in it the structures describing the options of interest, and set the opt.len field to the size of the buffer.

On output, as part of the ret parameter, the buffer contains the actual values of the options you described in the req parameter. You must allocate a buffer to hold the option information when the function returns and set the opt.maxlen field to the maximum length of this buffer. When the function returns, the opt.len field is set to the actual length of the buffer.

`flags`

For the req parameter, the flags field indicates the action to be taken as defined by the action flags enumeration (page 570). For the ret parameter, the flags field indicates the overall success or failure of the operation performed by the OTOptionManagement function, as defined by the "Open Transport Flags and Status Codes" (page 2702) enumeration.

Discussion
The option management structure is used for the req and ret parameters of the OTOptionManagement function. The req parameter is used to verify or negotiate option values. The ret parameter returns information about an endpoint's default, current, or negotiated values.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

TOTConfiguratorRef

```
typedef struct OpaqueTOTConfiguratorRef * TOTConfiguratorRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

TPortRecord

```
struct TPortRecord {
    OTLink fLink;
    char *fPortName;
    char *fModuleName;
    char *fResourceInfo;
    char *fSlotID;
    TPortRecord *fAlias;
    ItemCount fNumChildren;
    OTPortRef *fChildPorts;
    UInt32 fPortFlags;
    UInt32 fInfoFlags;
    UInt32 fCapabilities;
    OTPortRef fRef;
    streamtab *fStreamtab;
    void *fContext;
    void *fExtra;
};
```

Fields

trace_ids

```
struct trace_ids {
    short ti_mid;
    short ti_sid;
    char ti_level;
};
typedef struct trace_ids trace_ids;
```

Fields

`ti_mid`

`ti_sid`

`ti_level`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransportProtocol.h`

TRegisterReply

Stores information returned by the `OTRegisterName` function.

```
struct TRegisterReply {
    TNetbuf addr;
    OTNameID nameid;
};
typedef struct TRegisterReply TRegisterReply;
```

Fields

`addr`

A TNetbuf structure that specifies the location and size of a buffer containing the actual address of the entity whose name you have just registered. This information is passed back to you when the OTRegisterName function returns. You must allocate a buffer, set the addr.buf field to point to it, and set the addr.maxlen field to the size of the buffer.

`nameid`

A unique identifier passed to you when the OTRegisterName function returns. You can use this identifier when you call the OTDeleteNameByID function to delete the name.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

TRegisterRequest

Specifies the entity name you want to register using the OTRegisterName function and, optionally, to specify its address.

```
struct TRegisterRequest {
    TNetbuf name;
    TNetbuf addr;
    OTFlags flags;
};
typedef struct TRegisterRequest TRegisterRequest;
```

Fields

`name`

A TNetbuf structure that specifies the location and size of a buffer containing the entity name you want to register. You must allocate a buffer that contains the name, set the name.buf field to point to that buffer, and set the name.len field to the length of the buffer.

`addr`

A TNetbuf structure that specifies the location and size of a buffer containing the address associated with the entity whose name you want to register. You must allocate a buffer that contains the address, set the addr.buf field to point to that buffer, and set the addr.len field to the length of the buffer. The actual address with which the entity is associated is returned in the addr field of the TRegisterReply structure.

You can set the addr.len field to 0, in which case the underlying protocol finds an appropriate address to associate with the newly registered entity name.

`flags`

A field used to control registration. Normally, this field is set to 0 for default registration behavior. See the documentation for the naming service you are using for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

TReply

```
struct TReply {
    TNetbuf data;
    TNetbuf opt;
    OTSequence sequence;
};
typedef struct TReply TReply;
```

Fields
`data`
`opt`
`sequence`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

TRequest

```
struct TRequest {
    TNetbuf data;
    TNetbuf opt;
    OTSequence sequence;
};
typedef struct TRequest TRequest;
```

Fields
`data`
`opt`
`sequence`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

TUDErr

In the event of failure of the `OTSndUData` function, points to information that explains why it failed.function (page 462) has failed.

```
struct TUDErr {
    TNetbuf addr;
    TNetbuf opt;
    SInt32 error;
};
typedef struct TUDErr TUDErr;
```

Fields

`addr`

A `TNetbuf` (page 2546) structure that contains information about the destination address of the data sent using the `OTSndUData` (page 2400) function. The `OTRcvUData` (page 2381) function fills in the buffer referenced by this structure when the function returns. You must allocate a buffer to contain the address, initialize the `addr.buf` field to point to it, and set the `addr.maxlen` field to specify its maximum size. If you are not interested in address information, set `addr.maxlen` to 0.

`opt`

A `TNetbuf` (page 2546) structure that contains information about the options associated with the data sent using the `OTSndUData` (page 2400)function. The `OTRcvUDErr` (page 2382) function fills in the buffer referenced by this structure when the function returns. If you want to know this information, you must allocate a buffer to contain the option data, initialize the `opt.buf` field to point to it, and initialize the `opt.maxlen` field to specify the maximum size of the buffer. If you are not interested in option information, set the `opt.maxlen` field to 0.

`error`

On return, this specifies a protocol-dependent error code for the `OTSndUData` (page 2400) function that failed.

Discussion

In the event of failure of the `OTSndUData` (page 2400)function, points to information that explains why it failed.You pass this structure as a parameter to the `OTRcvUData` (page 2381) function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenTransport.h`

TUnitData

Describes the data being sent with the `OTSndUData` function and the data being read with the `OTRcvUData` function.function (page 467)

```
struct TUnitData {
    TNetbuf addr;
    TNetbuf opt;
    TNetbuf udata;
};
typedef struct TUnitData TUnitData;
```

Fields

`addr`

A `TNetbuf` structure for address information.

`opt`

A `TNetbuf` structure for option information.

`udata`
A `TNetbuf` structure for data.

Discussion
You use the `TUnitData` structure to describe the data being sent with the `OTSndUData` (page 2400) function and the data being read with the `OTRcvUData` (page 2381) function; you pass this structure as a parameter to each of these functions. When sending data you must initialize the `buf` and `len` fields of all the `TNetbuf` structures. When receiving data, you must initialize the `buf` and `maxlen` fields of all the `TNetbuf` structures.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

TUnitReply

```
struct TUnitReply {
    TNetbuf opt;
    TNetbuf udata;
    OTSequence sequence;
};
typedef struct TUnitReply TUnitReply;
```

Fields
`opt`
`udata`
`sequence`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

TUnitRequest

```
struct TUnitRequest {
    TNetbuf addr;
    TNetbuf opt;
    TNetbuf udata;
    OTSequence sequence;
};
typedef struct TUnitRequest TUnitRequest;
```

Fields

`addr`

`opt`

`udata`

`sequence`

Discussion

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

uchar_p

```
typedef UInt8 uchar_p;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenTransport.h`

uid_t

```
typedef UInt32 uid_t;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`types.h`

uint_t

```
typedef  uint_t;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

ushort_p

```
typedef UInt16 ushort_p;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OpenTransport.h

Constants

AF_8022

```
enum {
    AF_8022 = 8200
};
```

Constants
AF_8022
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProviders.h.

AF_ATALK_FAMILY

```
enum {
    AF_ATALK_FAMILY = 0x0100,
    AF_ATALK_DDP = 0x0100,
    AF_ATALK_DDPNBP = AF_ATALK_FAMILY + 1,
    AF_ATALK_NBP = AF_ATALK_FAMILY + 2,
    AF_ATALK_MNODE = AF_ATALK_FAMILY + 3
};
```

Constants
AF_ATALK_FAMILY
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProviders.h.

AF_ATALK_DDP
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProviders.h.

AF_ATALK_DDPNBP
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProviders.h.

`AF_ATALK_NBP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`AF_ATALK_MNODE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

AF_DNS

```
enum {
    AF_DNS = 42
};
```

Constants
`AF_DNS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

AF_INET

```
enum {
    AF_INET = 2
};
```

Constants
`AF_INET`

AF_ISDN

```
enum {
    AF_ISDN = 8192
};
```

Constants
`AF_ISDN`

ANYMARK

```
enum {
    ANYMARK = 0x01,
    LASTMARK = 0x02
};
```

Constants
`ANYMARK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`LASTMARK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

ATALK_IOC_FULLSELFSEND

```
enum {
    ATALK_IOC_FULLSELFSEND = ((MIOC_ATALK << 8) | 47),
    ADSP_IOC_FORWARDRESET = ((MIOC_ATALK << 8) | 60)
};
```

Constants

`ATALK_IOC_FULLSELFSEND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`ADSP_IOC_FORWARDRESET`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

ATK_DDP

```
enum {
    ATK_DDP = 'DDP ',
    ATK_AARP = 'AARP',
    ATK_ATP = 'ATP ',
    ATK_ADSP = 'ADSP',
    ATK_ASP = 'ASP ',
    ATK_PAP = 'PAP ',
    ATK_NBP = 'NBP ',
    ATK_ZIP = 'ZIP '
};
```

Constants

`ATK_DDP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`ATK_AARP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`ATK_ATP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`ATK_ADSP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`ATK_ASP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`ATK_PAP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`ATK_NBP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`ATK_ZIP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

BPRI_LO

```
enum {
    BPRI_LO = 1,
    BPRI_MED = 2,
    BPRI_HI = 3
};
```

Constants

`BPRI_LO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`BPRI_MED`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`BPRI_HI`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

CE_CONT

```
enum {
    CE_CONT = 0,
    CE_NOTE = 0,
    CE_WARN = 1,
    CE_PANIC = 2
};
```

Constants
`CE_CONT`
`CE_NOTE`
`CE_WARN`
`CE_PANIC`

CLONEOPEN

```
enum {
    CLONEOPEN = 0x02,
    MODOPEN = 0x01,
    OPENFAIL = -1
};
```

Constants
`CLONEOPEN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.
`MODOPEN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.
`OPENFAIL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

COM_ISDN

```
enum {
    COM_ISDN = 'ISDN'
};
```

Constants
`COM_ISDN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

COM_PPP

```
enum {
    COM_PPP = 'PPPC'
};
```

Constants

`COM_PPP`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

COM_SERIAL

```
enum {
    COM_SERIAL = 'SERL'
};
```

Constants

`COM_SERIAL`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

DDP_OPT_CHECKSUM

```
enum {
    DDP_OPT_CHECKSUM = 0x0600,
    DDP_OPT_SRCADDR = 0x2101,
    ATP_OPT_REPLYCNT = 0x2110,
    ATP_OPT_DATALEN = 0x2111,
    ATP_OPT_RELTIMER = 0x2112,
    ATP_OPT_TRANID = 0x2113,
    PAP_OPT_OPENRETRY = 0x2120
};
```

Constants

`DDP_OPT_CHECKSUM`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`DDP_OPT_SRCADDR`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`ATP_OPT_REPLYCNT`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`ATP_OPT_DATALEN`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`ATP_OPT_RELTIMER`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`ATP_OPT_TRANID`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`PAP_OPT_OPENRETRY`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

DDP_OPT_HOPCOUNT

```
enum {
    DDP_OPT_HOPCOUNT = 0x2100
};
```

Constants

`DDP_OPT_HOPCOUNT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

DL_ACCESS

```
enum {
    DL_ACCESS = 0x02,
    DL_BADADDR = 0x01,
    DL_BADCORR = 0x05,
    DL_BADDATA = 0x06,
    DL_BADPPA = 0x08,
    DL_BADPRIM = 0x09,
    DL_BADQOSPARAM = 0x0A,
    DL_BADQOSTYPE = 0x0B,
    DL_BADSAP = 0x00,
    DL_BADTOKEN = 0x0C,
    DL_BOUND = 0x0D,
    DL_INITFAILED = 0x0E,
    DL_NOADDR = 0x0F,
    DL_NOTINIT = 0x10,
    DL_OUTSTATE = 0x03,
    DL_SYSERR = 0x04,
    DL_UNSUPPORTED = 0x07,
    DL_UNDELIVERABLE = 0x11,
    DL_NOTSUPPORTED = 0x12,
    DL_TOOMANY = 0x13,
    DL_NOTENAB = 0x14,
    DL_BUSY = 0x15,
    DL_NOAUTO = 0x16,
    DL_NOXIDAUTO = 0x17,
    DL_NOTESTAUTO = 0x18,
    DL_XIDAUTO = 0x19,
    DL_TESTAUTO = 0x1A,
    DL_PENDING = 0x1B
};
```

Constants

`DL_ACCESS`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`DL_BADADDR`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`DL_BADCORR`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`DL_BADDATA`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`DL_BADPPA`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`DL_BADPRIM`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`DL_BADQOSPARAM`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_BADQOSTYPE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_BADSAP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_BADTOKEN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_BOUND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_INITFAILED`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_NOADDR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_NOTINIT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_OUTSTATE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_SYSERR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_UNSUPPORTED`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_UNDELIVERABLE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_NOTSUPPORTED`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_TOOMANY`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_NOTENAB`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_BUSY`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_NOAUTO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_NOXIDAUTO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_NOTESTAUTO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_XIDAUTO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_TESTAUTO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_PENDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

DL_AUTO_XID

```
enum {
    DL_AUTO_XID = 0x01,
    DL_AUTO_TEST = 0x02
};
```

Constants

`DL_AUTO_XID`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_AUTO_TEST`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

DL_CMD_MASK

```
enum {
    DL_CMD_MASK = 0x0F,
    DL_CMD_OK = 0x00,
    DL_CMD_RS = 0x01,
    DL_CMD_UE = 0x05,
    DL_CMD_PE = 0x06,
    DL_CMD_IP = 0x07,
    DL_CMD_UN = 0x09,
    DL_CMD_IT = 0x0F,
    DL_RSP_MASK = 0xF0,
    DL_RSP_OK = 0x00,
    DL_RSP_RS = 0x10,
    DL_RSP_NE = 0x30,
    DL_RSP_NR = 0x40,
    DL_RSP_UE = 0x50,
    DL_RSP_IP = 0x70,
    DL_RSP_UN = 0x90,
    DL_RSP_IT = 0xF0
};
```

Constants

`DL_CMD_MASK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CMD_OK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CMD_RS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CMD_UE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CMD_PE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CMD_IP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CMD_UN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CMD_IT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RSP_MASK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RSP_OK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RSP_RS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RSP_NE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RSP_NR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RSP_UE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RSP_IP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RSP_UN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RSP_IT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

DL_CODLS

```
enum {
    DL_CODLS = 0x01,
    DL_CLDLS = 0x02,
    DL_ACLDLS = 0x04
};
```

Constants

`DL_CODLS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CLDLS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_ACLDLS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

DL_CONREJ_DEST_UNKNOWN

```
enum {
    DL_CONREJ_DEST_UNKNOWN = 0x0800,
    DL_CONREJ_DEST_UNREACH_PERMANENT = 0x0801,
    DL_CONREJ_DEST_UNREACH_TRANSIENT = 0x0802,
    DL_CONREJ_QOS_UNAVAIL_PERMANENT = 0x0803,
    DL_CONREJ_QOS_UNAVAIL_TRANSIENT = 0x0804,
    DL_CONREJ_PERMANENT_COND = 0x0805,
    DL_CONREJ_TRANSIENT_COND = 0x0806,
    DL_DISC_ABNORMAL_CONDITION = 0x0807,
    DL_DISC_NORMAL_CONDITION = 0x0808,
    DL_DISC_PERMANENT_CONDITION = 0x0809,
    DL_DISC_TRANSIENT_CONDITION = 0x080A,
    DL_DISC_UNSPECIFIED = 0x080B
};
```

Constants

`DL_CONREJ_DEST_UNKNOWN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CONREJ_DEST_UNREACH_PERMANENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CONREJ_DEST_UNREACH_TRANSIENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CONREJ_QOS_UNAVAIL_PERMANENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CONREJ_QOS_UNAVAIL_TRANSIENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CONREJ_PERMANENT_COND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CONREJ_TRANSIENT_COND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DISC_ABNORMAL_CONDITION`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DISC_NORMAL_CONDITION`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DISC_PERMANENT_CONDITION`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DISC_TRANSIENT_CONDITION`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DISC_UNSPECIFIED`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

DL_CSMACD

```
enum {
    DL_CSMACD = 0x00,
    DL_TPB = 0x01,
    DL_TPR = 0x02,
    DL_METRO = 0x03,
    DL_ETHER = 0x04,
    DL_HDLC = 0x05,
    DL_CHAR = 0x06,
    DL_CTCA = 0x07,
    DL_FDDI = 0x08,
    DL_OTHER = 0x09
};
```

Constants

`DL_CSMACD`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_TPB`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_TPR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_METRO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_ETHER`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_HDLC`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CHAR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CTCA`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_FDDI`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_OTHER`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

DL_CURRENT_VERSION

```
enum {
    DL_CURRENT_VERSION = 0x02,
    DL_VERSION_2 = 0x02
};
```

Constants

`DL_CURRENT_VERSION`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_VERSION_2`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

DL_FACT_PHYS_ADDR

```
enum {
    DL_FACT_PHYS_ADDR = 0x01,
    DL_CURR_PHYS_ADDR = 0x02
};
```

Constants

`DL_FACT_PHYS_ADDR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CURR_PHYS_ADDR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

DL_INFO_REQ

```
enum {
    DL_INFO_REQ = 0x00,
    DL_INFO_ACK = 0x03,
    DL_ATTACH_REQ = 0x0B,
    DL_DETACH_REQ = 0x0C,
    DL_BIND_REQ = 0x01,
    DL_BIND_ACK = 0x04,
    DL_UNBIND_REQ = 0x02,
    DL_OK_ACK = 0x06,
    DL_ERROR_ACK = 0x05,
    DL_SUBS_BIND_REQ = 0x1B,
    DL_SUBS_BIND_ACK = 0x1C,
    DL_SUBS_UNBIND_REQ = 0x15,
    DL_ENABMULTI_REQ = 0x1D,
    DL_DISABMULTI_REQ = 0x1E,
    DL_PROMISCON_REQ = 0x1F,
    DL_PROMISCOFF_REQ = 0x20,
    DL_UNITDATA_REQ = 0x07,
    DL_UNITDATA_IND = 0x08,
    DL_UDERROR_IND = 0x09,
    DL_UDQOS_REQ = 0x0A,
    DL_CONNECT_REQ = 0x0D,
    DL_CONNECT_IND = 0x0E,
    DL_CONNECT_RES = 0x0F,
    DL_CONNECT_CON = 0x10,
    DL_TOKEN_REQ = 0x11,
    DL_TOKEN_ACK = 0x12,
    DL_DISCONNECT_REQ = 0x13,
    DL_DISCONNECT_IND = 0x14,
    DL_RESET_REQ = 0x17,
    DL_RESET_IND = 0x18,
    DL_RESET_RES = 0x19,
    DL_RESET_CON = 0x1A,
    DL_DATA_ACK_REQ = 0x21,
    DL_DATA_ACK_IND = 0x22,
    DL_DATA_ACK_STATUS_IND = 0x23,
    DL_REPLY_REQ = 0x24,
    DL_REPLY_IND = 0x25,
    DL_REPLY_STATUS_IND = 0x26,
    DL_REPLY_UPDATE_REQ = 0x27,
    DL_REPLY_UPDATE_STATUS_IND = 0x28,
    DL_XID_REQ = 0x29,
    DL_XID_IND = 0x2A,
    DL_XID_RES = 0x2B,
    DL_XID_CON = 0x2C,
    DL_TEST_REQ = 0x2D,
    DL_TEST_IND = 0x2E,
    DL_TEST_RES = 0x2F,
    DL_TEST_CON = 0x30,
    DL_PHYS_ADDR_REQ = 0x31,
    DL_PHYS_ADDR_ACK = 0x32,
    DL_SET_PHYS_ADDR_REQ = 0x33,
    DL_GET_STATISTICS_REQ = 0x34,
    DL_GET_STATISTICS_ACK = 0x35
};
```

Constants

`DL_INFO_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_INFO_ACK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_ATTACH_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DETACH_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_BIND_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_BIND_ACK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_UNBIND_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_OK_ACK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_ERROR_ACK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_SUBS_BIND_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_SUBS_BIND_ACK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_SUBS_UNBIND_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_ENABMULTI_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DISABMULTI_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_PROMISCON_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_PROMISCOFF_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_UNITDATA_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_UNITDATA_IND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_UDERROR_IND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_UDQOS_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CONNECT_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CONNECT_IND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CONNECT_RES`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CONNECT_CON`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_TOKEN_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_TOKEN_ACK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DISCONNECT_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DISCONNECT_IND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RESET_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RESET_IND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RESET_RES`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RESET_CON`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DATA_ACK_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DATA_ACK_IND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DATA_ACK_STATUS_IND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_REPLY_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_REPLY_IND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_REPLY_STATUS_IND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_REPLY_UPDATE_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_REPLY_UPDATE_STATUS_IND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_XID_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_XID_IND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_XID_RES`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_XID_CON`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_TEST_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_TEST_IND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_TEST_RES`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_TEST_CON`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_PHYS_ADDR_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_PHYS_ADDR_ACK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_SET_PHYS_ADDR_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_GET_STATISTICS_REQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_GET_STATISTICS_ACK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

DL_INFO_REQ_SIZE

```
enum {
    DL_INFO_REQ_SIZE = sizeof(dl_info_req_t),
    DL_INFO_ACK_SIZE = sizeof(dl_info_ack_t),
    DL_ATTACH_REQ_SIZE = sizeof(dl_attach_req_t),
    DL_DETACH_REQ_SIZE = sizeof(dl_detach_req_t),
    DL_BIND_REQ_SIZE = sizeof(dl_bind_req_t),
    DL_BIND_ACK_SIZE = sizeof(dl_bind_ack_t),
    DL_UNBIND_REQ_SIZE = sizeof(dl_unbind_req_t),
    DL_SUBS_BIND_REQ_SIZE = sizeof(dl_subs_bind_req_t),
    DL_SUBS_BIND_ACK_SIZE = sizeof(dl_subs_bind_ack_t),
    DL_SUBS_UNBIND_REQ_SIZE = sizeof(dl_subs_unbind_req_t),
    DL_OK_ACK_SIZE = sizeof(dl_ok_ack_t),
    DL_ERROR_ACK_SIZE = sizeof(dl_error_ack_t),
    DL_CONNECT_REQ_SIZE = sizeof(dl_connect_req_t),
    DL_CONNECT_IND_SIZE = sizeof(dl_connect_ind_t),
    DL_CONNECT_RES_SIZE = sizeof(dl_connect_res_t),
    DL_CONNECT_CON_SIZE = sizeof(dl_connect_con_t),
    DL_TOKEN_REQ_SIZE = sizeof(dl_token_req_t),
    DL_TOKEN_ACK_SIZE = sizeof(dl_token_ack_t),
    DL_DISCONNECT_REQ_SIZE = sizeof(dl_disconnect_req_t),
    DL_DISCONNECT_IND_SIZE = sizeof(dl_disconnect_ind_t),
    DL_RESET_REQ_SIZE = sizeof(dl_reset_req_t),
    DL_RESET_IND_SIZE = sizeof(dl_reset_ind_t),
    DL_RESET_RES_SIZE = sizeof(dl_reset_res_t),
    DL_RESET_CON_SIZE = sizeof(dl_reset_con_t),
    DL_UNITDATA_REQ_SIZE = sizeof(dl_unitdata_req_t),
    DL_UNITDATA_IND_SIZE = sizeof(dl_unitdata_ind_t),
    DL_UDERROR_IND_SIZE = sizeof(dl_uderror_ind_t),
    DL_UDQOS_REQ_SIZE = sizeof(dl_udqos_req_t),
    DL_ENABMULTI_REQ_SIZE = sizeof(dl_enabmulti_req_t),
    DL_DISABMULTI_REQ_SIZE = sizeof(dl_disabmulti_req_t),
    DL_PROMISCON_REQ_SIZE = sizeof(dl_promiscon_req_t),
    DL_PROMISCOFF_REQ_SIZE = sizeof(dl_promiscoff_req_t),
    DL_PHYS_ADDR_REQ_SIZE = sizeof(dl_phys_addr_req_t),
    DL_PHYS_ADDR_ACK_SIZE = sizeof(dl_phys_addr_ack_t),
    DL_SET_PHYS_ADDR_REQ_SIZE = sizeof(dl_set_phys_addr_req_t),
    DL_GET_STATISTICS_REQ_SIZE = sizeof(dl_get_statistics_req_t),
    DL_GET_STATISTICS_ACK_SIZE = sizeof(dl_get_statistics_ack_t),
    DL_XID_REQ_SIZE = sizeof(dl_xid_req_t),
    DL_XID_IND_SIZE = sizeof(dl_xid_ind_t),
    DL_XID_RES_SIZE = sizeof(dl_xid_res_t),
    DL_XID_CON_SIZE = sizeof(dl_xid_con_t),
    DL_TEST_REQ_SIZE = sizeof(dl_test_req_t),
    DL_TEST_IND_SIZE = sizeof(dl_test_ind_t),
    DL_TEST_RES_SIZE = sizeof(dl_test_res_t),
    DL_TEST_CON_SIZE = sizeof(dl_test_con_t),
    DL_DATA_ACK_REQ_SIZE = sizeof(dl_data_ack_req_t),
    DL_DATA_ACK_IND_SIZE = sizeof(dl_data_ack_ind_t),
    DL_DATA_ACK_STATUS_IND_SIZE = sizeof(dl_data_ack_status_ind_t),
    DL_REPLY_REQ_SIZE = sizeof(dl_reply_req_t),
    DL_REPLY_IND_SIZE = sizeof(dl_reply_ind_t),
    DL_REPLY_STATUS_IND_SIZE = sizeof(dl_reply_status_ind_t),
    DL_REPLY_UPDATE_REQ_SIZE = sizeof(dl_reply_update_req_t),
    DL_REPLY_UPDATE_STATUS_IND_SIZE = sizeof(dl_reply_update_status_ind_t)
};
```

Constants

`DL_INFO_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_INFO_ACK_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_ATTACH_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DETACH_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_BIND_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_BIND_ACK_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_UNBIND_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_SUBS_BIND_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_SUBS_BIND_ACK_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_SUBS_UNBIND_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_OK_ACK_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_ERROR_ACK_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CONNECT_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CONNECT_IND_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CONNECT_RES_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CONNECT_CON_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_TOKEN_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_TOKEN_ACK_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DISCONNECT_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DISCONNECT_IND_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RESET_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RESET_IND_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RESET_RES_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RESET_CON_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_UNITDATA_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_UNITDATA_IND_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_UDERROR_IND_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_UDQOS_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_ENABMULTI_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DISABMULTI_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_PROMISCON_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_PROMISCOFF_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_PHYS_ADDR_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_PHYS_ADDR_ACK_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_SET_PHYS_ADDR_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_GET_STATISTICS_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_GET_STATISTICS_ACK_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_XID_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_XID_IND_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_XID_RES_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_XID_CON_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_TEST_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_TEST_IND_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_TEST_RES_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_TEST_CON_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DATA_ACK_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DATA_ACK_IND_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DATA_ACK_STATUS_IND_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_REPLY_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_REPLY_IND_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_REPLY_STATUS_IND_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_REPLY_UPDATE_REQ_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_REPLY_UPDATE_STATUS_IND_SIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

DL_IOC_HDR_INFO

```
enum {
    DL_IOC_HDR_INFO = ((MIOC_DLPI << 8) | 10)
};
```

Constants

`DL_IOC_HDR_INFO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

DL_NONE

```
enum {
    DL_NONE = 0x0B01,
    DL_MONITOR = 0x0B02,
    DL_MAXIMUM = 0x0B03
};
```

Constants

`DL_NONE`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`DL_MONITOR`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`DL_MAXIMUM`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

DL_PEER_BIND

```
enum {
    DL_PEER_BIND = 0x01,
    DL_HIERARCHICAL_BIND = 0x02
};
```

Constants

`DL_PEER_BIND`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`DL_HIERARCHICAL_BIND`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

DL_POLL_FINAL

```
enum {
    DL_POLL_FINAL = 0x01
};
```

Constants

`DL_POLL_FINAL`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

DL_PROMISC_OFF

```
enum {
    DL_PROMISC_OFF = 0
};
```

Constants

`DL_PROMISC_OFF`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

DL_PROMISC_PHYS

```
enum {
    DL_PROMISC_PHYS = 0x01,
    DL_PROMISC_SAP = 0x02,
    DL_PROMISC_MULTI = 0x03
};
```

Constants

`DL_PROMISC_PHYS`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`DL_PROMISC_SAP`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`DL_PROMISC_MULTI`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

DL_PROVIDER

```
enum {
    DL_PROVIDER = 0x0700,
    DL_USER = 0x0701
};
```

Constants

`DL_PROVIDER`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`DL_USER`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

DL_QOS_CO_RANGE1

```
enum {
    DL_QOS_CO_RANGE1 = 0x0101,
    DL_QOS_CO_SEL1 = 0x0102,
    DL_QOS_CL_RANGE1 = 0x0103,
    DL_QOS_CL_SEL1 = 0x0104
};
```

Constants

`DL_QOS_CO_RANGE1`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_QOS_CO_SEL1`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_QOS_CL_RANGE1`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_QOS_CL_SEL1`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

DL_RESET_FLOW_CONTROL

```
enum {
    DL_RESET_FLOW_CONTROL = 0x0900,
    DL_RESET_LINK_ERROR = 0x0901,
    DL_RESET_RESYNCH = 0x0902
};
```

Constants

`DL_RESET_FLOW_CONTROL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RESET_LINK_ERROR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RESET_RESYNCH`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

DL_RQST_RSP

```
enum {
    DL_RQST_RSP = 0x01,
    DL_RQST_NORSP = 0x02
};
```

Constants

`DL_RQST_RSP`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`DL_RQST_NORSP`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

DL_STYLE1

```
enum {
    DL_STYLE1 = 0x0500,
    DL_STYLE2 = 0x0501
};
```

Constants

`DL_STYLE1`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`DL_STYLE2`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

DL_UNATTACHED

```
enum {
    DL_UNATTACHED = 0x04,
    DL_ATTACH_PENDING = 0x05,
    DL_DETACH_PENDING = 0x06,
    DL_UNBOUND = 0x00,
    DL_BIND_PENDING = 0x01,
    DL_UNBIND_PENDING = 0x02,
    DL_IDLE = 0x03,
    DL_UDQOS_PENDING = 0x07,
    DL_OUTCON_PENDING = 0x08,
    DL_INCON_PENDING = 0x09,
    DL_CONN_RES_PENDING = 0x0A,
    DL_DATAXFER = 0x0B,
    DL_USER_RESET_PENDING = 0x0C,
    DL_PROV_RESET_PENDING = 0x0D,
    DL_RESET_RES_PENDING = 0x0E,
    DL_DISCON8_PENDING = 0x0F,
    DL_DISCON9_PENDING = 0x10,
    DL_DISCON11_PENDING = 0x11,
    DL_DISCON12_PENDING = 0x12,
    DL_DISCON13_PENDING = 0x13,
    DL_SUBS_BIND_PND = 0x14,
    DL_SUBS_UNBIND_PND = 0x15
};
```

Constants

`DL_UNATTACHED`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_ATTACH_PENDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DETACH_PENDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_UNBOUND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_BIND_PENDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_UNBIND_PENDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_IDLE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_UDQOS_PENDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_OUTCON_PENDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_INCON_PENDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_CONN_RES_PENDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DATAXFER`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_USER_RESET_PENDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_PROV_RESET_PENDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_RESET_RES_PENDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DISCON8_PENDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DISCON9_PENDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DISCON11_PENDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DISCON12_PENDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_DISCON13_PENDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_SUBS_BIND_PND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_SUBS_UNBIND_PND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

DL_UNKNOWN

```
enum {
    DL_UNKNOWN = -1,
    DL_QOS_DONT_CARE = -2
};
```

Constants

`DL_UNKNOWN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`DL_QOS_DONT_CARE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

DVMRP_INIT

```
enum {
    DVMRP_INIT = 100,
    DVMRP_DONE = 101,
    DVMRP_ADD_VIF = 102,
    DVMRP_DEL_VIF = 103,
    DVMRP_ADD_LGRP = 104,
    DVMRP_DEL_LGRP = 105,
    DVMRP_ADD_MRT = 106,
    DVMRP_DEL_MRT = 107
};
```

Constants

`DVMRP_INIT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`DVMRP_DONE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`DVMRP_ADD_VIF`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`DVMRP_DEL_VIF`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`DVMRP_ADD_LGRP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`DVMRP_DEL_LGRP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`DVMRP_ADD_MRT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`DVMRP_DEL_MRT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

EAddrType

```
typedef UInt32 EAddrType;
enum {
    keaStandardAddress = 0,
    keaMulticast = 1,
    keaBroadcast = 2,
    keaBadAddress = 3,
    keaRawPacketBit = 0x80000000,
    keaTimeStampBit = 0x40000000
};
```

Constants

`keaStandardAddress`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`keaMulticast`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`keaBroadcast`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`keaBadAddress`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`keaRawPacketBit`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`keaTimeStampBit`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

EPERM

```
enum {
    EPERM = 1,
    ENOENT = 2,
    ENORSRC = 3,
    EINTR = 4,
    EIO = 5,
    ENXIO = 6,
    EBADF = 9,
    EAGAIN = 11,
    ENOMEM = 12,
    EACCES = 13,
    EFAULT = 14,
    EBUSY = 16,
    EEXIST = 17,
    ENODEV = 19,
    EINVAL = 22,
    ENOTTY = 25,
    EPIPE = 32,
    ERANGE = 34,
    EDEADLK = 35,
    EWOULDBLOCK = 35,
    EALREADY = 37,
    ENOTSOCK = 38,
    EDESTADDRREQ = 39,
    EMSGSIZE = 40,
    EPROTOTYPE = 41,
    ENOPROTOOPT = 42,
    EPROTONOSUPPORT = 43,
    ESOCKTNOSUPPORT = 44,
    EOPNOTSUPP = 45,
    EADDRINUSE = 48,
    EADDRNOTAVAIL = 49,
    ENETDOWN = 50,
    ENETUNREACH = 51,
    ENETRESET = 52,
    ECONNABORTED = 53,
    ECONNRESET = 54,
    ENOBUFS = 55,
    EISCONN = 56,
    ENOTCONN = 57,
    ESHUTDOWN = 58,
    ETOOMANYREFS = 59,
    ETIMEDOUT = 60,
    ECONNREFUSED = 61,
    EHOSTDOWN = 64,
    EHOSTUNREACH = 65,
    EPROTO = 70,
    ETIME = 71,
    ENOSR = 72,
    EBADMSG = 73,
    ECANCEL = 74,
    ENOSTR = 75,
    ENODATA = 76,
    EINPROGRESS = 77,
    ESRCH = 78,
    ENOMSG = 79,
```

```
    ELASTERRNO = 79
};
```

Constants

`EPERM`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ENOENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ENORSRC`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EINTR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EIO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ENXIO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EBADF`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EAGAIN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ENOMEM`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EACCES`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EFAULT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EBUSY`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EEXIST`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ENODEV`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EINVAL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ENOTTY`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EPIPE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ERANGE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EDEADLK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EWOULDBLOCK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EALREADY`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ENOTSOCK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EDESTADDRREQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EMSGSIZE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EPROTOTYPE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ENOPROTOOPT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EPROTONOSUPPORT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ESOCKTNOSUPPORT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EOPNOTSUPP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EADDRINUSE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EADDRNOTAVAIL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ENETDOWN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ENETUNREACH`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ENETRESET`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ECONNABORTED`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ECONNRESET`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ENOBUFS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EISCONN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ENOTCONN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ESHUTDOWN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ETOOMANYREFS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ETIMEDOUT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ECONNREFUSED`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EHOSTDOWN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EHOSTUNREACH`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EPROTO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ETIME`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ENOSR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EBADMSG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ECANCEL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ENOSTR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ENODATA`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`EINPROGRESS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ESRCH`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ENOMSG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`ELASTERRNO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

FLUSHALL

```
enum {
    FLUSHALL = 1,
    FLUSHDATA = 0
};
```

Constants

`FLUSHALL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`FLUSHDATA`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

FLUSHR

```
enum {
    FLUSHR = 0x01,
    FLUSHW = 0x02,
    FLUSHRW = (FLUSHW | FLUSHR)
};
```

Constants

`FLUSHR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`FLUSHW`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`FLUSHRW`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

FMNAMESZ

```
enum {
    FMNAMESZ = 31
};
```

Constants

`FMNAMESZ`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

I_NREAD

```
enum {
    I_NREAD = ((MIOC_STREAMIO << 8) | 1),
    I_PUSH = ((MIOC_STREAMIO << 8) | 2),
    I_POP = ((MIOC_STREAMIO << 8) | 3),
    I_LOOK = ((MIOC_STREAMIO << 8) | 4),
    I_FLUSH = ((MIOC_STREAMIO << 8) | 5),
    I_SRDOPT = ((MIOC_STREAMIO << 8) | 6),
    I_GRDOPT = ((MIOC_STREAMIO << 8) | 7),
    I_STR = ((MIOC_STREAMIO << 8) | 8),
    I_SETSIG = ((MIOC_STREAMIO << 8) | 9),
    I_GETSIG = ((MIOC_STREAMIO << 8) | 10),
    I_FIND = ((MIOC_STREAMIO << 8) | 11),
    I_LINK = ((MIOC_STREAMIO << 8) | 12),
    I_UNLINK = ((MIOC_STREAMIO << 8) | 13),
    I_PEEK = ((MIOC_STREAMIO << 8) | 15),
    I_FDINSERT = ((MIOC_STREAMIO << 8) | 16),
    I_SENDFD = ((MIOC_STREAMIO << 8) | 17),
    I_RECVFD = ((MIOC_STREAMIO << 8) | 18),
    I_FLUSHBAND = ((MIOC_STREAMIO << 8) | 19),
    I_SWROPT = ((MIOC_STREAMIO << 8) | 20),
    I_GWROPT = ((MIOC_STREAMIO << 8) | 21),
    I_LIST = ((MIOC_STREAMIO << 8) | 22),
    I_ATMARK = ((MIOC_STREAMIO << 8) | 23),
    I_CKBAND = ((MIOC_STREAMIO << 8) | 24),
    I_GETBAND = ((MIOC_STREAMIO << 8) | 25),
    I_CANPUT = ((MIOC_STREAMIO << 8) | 26),
    I_SETCLTIME = ((MIOC_STREAMIO << 8) | 27),
    I_GETCLTIME = ((MIOC_STREAMIO << 8) | 28),
    I_PLINK = ((MIOC_STREAMIO << 8) | 29),
    I_PUNLINK = ((MIOC_STREAMIO << 8) | 30),
    I_GETMSG = ((MIOC_STREAMIO << 8) | 40),
    I_PUTMSG = ((MIOC_STREAMIO << 8) | 41),
    I_POLL = ((MIOC_STREAMIO << 8) | 42),
    I_SETDELAY = ((MIOC_STREAMIO << 8) | 43),
    I_GETDELAY = ((MIOC_STREAMIO << 8) | 44),
    I_RUN_QUEUES = ((MIOC_STREAMIO << 8) | 45),
    I_GETPMSG = ((MIOC_STREAMIO << 8) | 46),
    I_PUTPMSG = ((MIOC_STREAMIO << 8) | 47),
    I_AUTOPUSH = ((MIOC_STREAMIO << 8) | 48),
    I_PIPE = ((MIOC_STREAMIO << 8) | 49),
    I_HEAP_REPORT = ((MIOC_STREAMIO << 8) | 50),
    I_FIFO = ((MIOC_STREAMIO << 8) | 51)
};
```

Constants

`I_NREAD`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`I_PUSH`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`I_POP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_LOOK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_FLUSH`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_SRDOPT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_GRDOPT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_STR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_SETSIG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_GETSIG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_FIND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_LINK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_UNLINK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_PEEK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_FDINSERT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_SENDFD`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_RECVFD`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_FLUSHBAND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_SWROPT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_GWROPT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_LIST`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_ATMARK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_CKBAND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_GETBAND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_CANPUT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_SETCLTIME`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_GETCLTIME`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_PLINK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_PUNLINK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_GETMSG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_PUTMSG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_POLL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_SETDELAY`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_GETDELAY`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_RUN_QUEUES`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_GETPMSG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_PUTPMSG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_AUTOPUSH`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_PIPE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_HEAP_REPORT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_FIFO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

I_OTGetMiscellaneousEvents

```
enum {
    I_OTGetMiscellaneousEvents = ((MIOC_OT << 8) | 1),
    I_OTSetFramingType = ((MIOC_OT << 8) | 2),
    kOTGetFramingValue = 0xFFFFFFFF,
    I_OTSetRawMode = ((MIOC_OT << 8) | 3),
    kOTSetRecvMode = 0x01,
    kOTSendErrorPacket = 0x02,
    I_OTConnect = ((MIOC_OT << 8) | 4),
    I_OTDisconnect = ((MIOC_OT << 8) | 5),
    I_OTScript = ((MIOC_OT << 8) | 6)
};
```

Constants

`I_OTGetMiscellaneousEvents`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`I_OTSetFramingType`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTGetFramingValue`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`I_OTSetRawMode`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTSetRecvMode`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTSendErrorPacket`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`I_OTConnect`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`I_OTDisconnect`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`I_OTScript`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

I_OTISDNAlerting

```
enum {
    I_OTISDNAlerting = ((MIOC_ISDN << 8) | 100),
    I_OTISDNSuspend = ((MIOC_ISDN << 8) | 101),
    I_OTISDNSuspendAcknowledge = ((MIOC_ISDN << 8) | 102),
    I_OTISDNSuspendReject = ((MIOC_ISDN << 8) | 103),
    I_OTISDNResume = ((MIOC_ISDN << 8) | 104),
    I_OTISDNResumeAcknowledge = ((MIOC_ISDN << 8) | 105),
    I_OTISDNResumeReject = ((MIOC_ISDN << 8) | 106),
    I_OTISDNFaciltity = ((MIOC_ISDN << 8) | 107)
};
```

Constants

`I_OTISDNAlerting`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`I_OTISDNSuspend`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`I_OTISDNSuspendAcknowledge`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`I_OTISDNSuspendReject`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`I_OTISDNResume`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`I_OTISDNResumeAcknowledge`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`I_OTISDNResumeReject`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`I_OTISDNFaciltity`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

I_SAD_SAP

```
enum {
    I_SAD_SAP = ((MIOC_SAD << 8) | 1),
    I_SAD_GAP = ((MIOC_SAD << 8) | 2),
    I_SAD_VML = ((MIOC_SAD << 8) | 3)
};
```

Constants

`I_SAD_SAP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_SAD_GAP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`I_SAD_VML`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

I_SetSerialDTR

```
enum {
    I_SetSerialDTR = ((MIOC_SRL << 8) | 0),
    kOTSerialSetDTROff = 0,
    kOTSerialSetDTROn = 1,
    I_SetSerialBreak = ((MIOC_SRL << 8) | 1),
    kOTSerialSetBreakOn = 0xFFFFFFFF,
    kOTSerialSetBreakOff = 0,
    I_SetSerialXOffState = ((MIOC_SRL << 8) | 2),
    kOTSerialForceXOffTrue = 1,
    kOTSerialForceXOffFalse = 0,
    I_SetSerialXOn = ((MIOC_SRL << 8) | 3),
    kOTSerialSendXOnAlways = 1,
    kOTSerialSendXOnIfXOffTrue = 0,
    I_SetSerialXOff = ((MIOC_SRL << 8) | 4),
    kOTSerialSendXOffAlways = 1,
    kOTSerialSendXOffIfXOnTrue = 0
};
```

Constants

`I_SetSerialDTR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialSetDTROff`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialSetDTROn`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`I_SetSerialBreak`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialSetBreakOn`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialSetBreakOff`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`I_SetSerialXOffState`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialForceXOffTrue`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialForceXOffFalse`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`I_SetSerialXOn`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialSendXOnAlways`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialSendXOnIfXOffTrue`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`I_SetSerialXOff`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialSendXOffAlways`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialSendXOffIfXOnTrue`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

I_TRCLOG

```
enum {
    I_TRCLOG = ((MIOC_STRLOG << 8) | 1),
    I_ERRLOG = ((MIOC_STRLOG << 8) | 2)
};
```

Constants

`I_TRCLOG`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`I_ERRLOG`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

INET_IP

```
enum {
    INET_IP = 0x00,
    INET_TCP = 0x06,
    INET_UDP = 0x11
};
```

Constants

`INET_IP`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`INET_TCP`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`INET_UDP`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

INFPSZ

```
enum {
    INFPSZ = -1
};
```

Constants

`INFPSZ`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

INFTIM

```
enum {
    INFTIM = 0xFFFFFFFF
};
```

Constants

`INFTIM`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

IP_OPTIONS

```
enum {
    IP_OPTIONS = 1,
    IP_TOS = 2,
    IP_TTL = 3,
    IP_REUSEADDR = 4,
    IP_DONTROUTE = 16,
    IP_BROADCAST = 32,
    IP_REUSEPORT = 512,
    IP_HDRINCL = 4098,
    IP_RCVOPTS = 4101,
    IP_RCVDSTADDR = 4103,
    IP_MULTICAST_IF = 4112,
    IP_MULTICAST_TTL = 4113,
    IP_MULTICAST_LOOP = 4114,
    IP_ADD_MEMBERSHIP = 4115,
    IP_DROP_MEMBERSHIP = 4116,
    IP_BROADCAST_IFNAME = 4117,
    IP_RCVIFADDR = 4118
};
```

Constants

```
IP_OPTIONS
IP_TOS
IP_TTL
IP_REUSEADDR
IP_DONTROUTE
IP_BROADCAST
IP_REUSEPORT
IP_HDRINCL
IP_RCVOPTS
IP_RCVDSTADDR
IP_MULTICAST_IF
IP_MULTICAST_TTL
IP_MULTICAST_LOOP
IP_ADD_MEMBERSHIP
IP_DROP_MEMBERSHIP
IP_BROADCAST_IFNAME
IP_RCVIFADDR
```

IPCP_OPT_GETREMOTEPROTOADDR

```
enum {
    IPCP_OPT_GETREMOTEPROTOADDR = 0x00007000,
    IPCP_OPT_GETLOCALPROTOADDR = 0x00007001,
    IPCP_OPT_TCPHDRCOMPRESSION = 0x00007002,
    LCP_OPT_PPPCOMPRESSION = 0x00007003,
    LCP_OPT_MRU = 0x00007004,
    LCP_OPT_RCACCMAP = 0x00007005,
    LCP_OPT_TXACCMAP = 0x00007006,
    SEC_OPT_OUTAUTHENTICATION = 0x00007007,
    SEC_OPT_ID = 0x00007008,
    SEC_OPT_PASSWORD = 0x00007009,
```

```
    CC_OPT_REMINDERTIMER = 0x00007010,
    CC_OPT_IPIDLETIMER = 0x00007011,
    CC_OPT_DTEADDRESSTYPE = 0x00007012,
    CC_OPT_DTEADDRESS = 0x00007013,
    CC_OPT_CALLINFO = 0x00007014,
    CC_OPT_GETMISCINFO = 0x00007015,
    PPP_OPT_GETCURRENTSTATE = 0x00007016,
    LCP_OPT_ECHO = 0x00007017,
    CC_OPT_SERIALPORTNAME = 0x00007200
};
```

Constants

`IPCP_OPT_GETREMOTEPROTOADDR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`IPCP_OPT_GETLOCALPROTOADDR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`IPCP_OPT_TCPHDRCOMPRESSION`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`LCP_OPT_PPPCOMPRESSION`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`LCP_OPT_MRU`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`LCP_OPT_RCACCMAP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`LCP_OPT_TXACCMAP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`SEC_OPT_OUTAUTHENTICATION`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`SEC_OPT_ID`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`SEC_OPT_PASSWORD`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`CC_OPT_REMINDERTIMER`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`CC_OPT_IPIDLETIMER`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`CC_OPT_DTEADDRESSTYPE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`CC_OPT_DTEADDRESS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`CC_OPT_CALLINFO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`CC_OPT_GETMISCINFO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`PPP_OPT_GETCURRENTSTATE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`LCP_OPT_ECHO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`CC_OPT_SERIALPORTNAME`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

ISDN_OPT_COMMTYPE

```
enum {
    ISDN_OPT_COMMTYPE = 0x0200,
    ISDN_OPT_FRAMINGTYPE = 0x0201,
    ISDN_OPT_56KADAPTATION = 0x0202
};
```

Constants

`ISDN_OPT_COMMTYPE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`ISDN_OPT_FRAMINGTYPE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`ISDN_OPT_56KADAPTATION`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

k8022BasicAddressLength

```
enum {
    k8022BasicAddressLength = sizeof(OTAddressType) + k48BitAddrLength
+ sizeof(UInt16),
    k8022SNAPAddressLength = sizeof(OTAddressType) + k48BitAddrLength
+ sizeof(UInt16) + k8022SNAPLength
};
```

Constants

`k8022BasicAddressLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`k8022SNAPAddressLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kAF_ISDN

```
enum {
    kAF_ISDN = 0x2000
};
```

Constants

`kAF_ISDN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kAllATalkRoutersDown

```
enum {
    kAllATalkRoutersDown = 0,
    kLocalATalkRoutersDown = -1L,
    kARARouterDisconnected = -2L
};
```

Constants

`kAllATalkRoutersDown`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kLocalATalkRoutersDown`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kARARouterDisconnected`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kAllDHCPOptions

```
enum {
    kAllDHCPOptions = -1,
    kDHCPLongOption = 126,
    kDHCPLongOptionReq = 127
};
```

Constants

`kAllDHCPOptions`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kDHCPLongOption`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kDHCPLongOptionReq`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kAppleTalkEvent

```
enum {
    kAppleTalkEvent = kPROTOCOLEVENT | 0x00010000,
    T_GETMYZONECOMPLETE = kAppleTalkEvent + 1,
    T_GETLOCALZONESCOMPLETE = kAppleTalkEvent + 2,
    T_GETZONELISTCOMPLETE = kAppleTalkEvent + 3,
    T_GETATALKINFOCOMPLETE = kAppleTalkEvent + 4,
    T_ATALKROUTERDOWNEVENT = kAppleTalkEvent + 51,
    T_ATALKROUTERUPEVENT = kAppleTalkEvent + 52,
    T_ATALKZONENAMECHANGEDEVENT = kAppleTalkEvent + 53,
    T_ATALKCONNECTIVITYCHANGEDEVENT = kAppleTalkEvent + 54,
    T_ATALKINTERNETAVAILABLEEVENT = kAppleTalkEvent + 55,
    T_ATALKCABLERANGECHANGEDEVENT = kAppleTalkEvent + 56
};
```

Constants

`kAppleTalkEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_GETMYZONECOMPLETE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_GETLOCALZONESCOMPLETE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_GETZONELISTCOMPLETE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_GETATALKINFOCOMPLETE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_ATALKROUTERDOWNEVENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_ATALKROUTERUPEVENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_ATALKZONENAMECHANGEDEVENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_ATALKCONNECTIVITYCHANGEDEVENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_ATALKINTERNETAVAILABLEEVENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_ATALKCABLERANGECHANGEDEVENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kARARouterOnline

```
enum {
    kARARouterOnline = -1L,
    kATalkRouterOnline = 0,
    kLocalATalkRouterOnline = -2L
};
```

Constants

`kARARouterOnline`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kATalkRouterOnline`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kLocalATalkRouterOnline`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kATalkInfoIsExtended

```
enum {
    kATalkInfoIsExtended = 0x0001,
    kATalkInfoHasRouter = 0x0002,
    kATalkInfoOneZone = 0x0004
};
```

Constants

`kATalkInfoIsExtended`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kATalkInfoHasRouter`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kATalkInfoOneZone`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kCCReminderTimerDisabled

```
enum {
    kCCReminderTimerDisabled = 0,
    kCCIPIdleTimerDisabled = 0
};
```

Constants

`kCCReminderTimerDisabled`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kCCIPIdleTimerDisabled`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kDDPAddressLength

```
enum {
    kDDPAddressLength = 8,
    kNBPAddressLength = kNBPEntityBufferSize,
    kAppleTalkAddressLength = kDDPAddressLength + kNBPEntityBufferSize
};
```

Constants

`kDDPAddressLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kNBPAddressLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kAppleTalkAddressLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kDefaultAppleTalkServicesPath

```
enum {
    kDefaultAppleTalkServicesPath = -3
};
```

Constants
`kDefaultAppleTalkServicesPath`

kDefaultInetInterface

```
enum {
    kDefaultInetInterface = -1
};
```

Constants
`kDefaultInetInterface`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kDefaultInternetServicesPath

```
enum {
    kDefaultInternetServicesPath = -3
};
```

Constants
`kDefaultInternetServicesPath`

kE164Address

```
enum {
    kE164Address = 1,
    kPhoneAddress = 1,
    kCompoundPhoneAddress = 2,
    kX121Address = 3
};
```

Constants
`kE164Address`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPhoneAddress`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kCompoundPhoneAddress`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kX121Address`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kECHO_TSDU

```
enum {
    kECHO_TSDU = 585
};
```

Constants
`kECHO_TSDU`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kEnetPacketHeaderLength

```
enum {
    kEnetPacketHeaderLength = (2 * k48BitAddrLength) + k8022DLSAPLength,
    kEnetTSDU = 1514,
    kTokenRingTSDU = 4458,
    kFDDITSDU = 4458,
    k8022SAPLength = 1,
    k8022BasicHeaderLength = 3,
    k8022SNAPHeaderLength = k8022SNAPLength + k8022BasicHeaderLength
};
```

Constants

`kEnetPacketHeaderLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kEnetTSDU`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kTokenRingTSDU`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kFDDITSDU`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`k8022SAPLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`k8022BasicHeaderLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`k8022SNAPHeaderLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kFirstMinorNumber

```
enum {
    kFirstMinorNumber = 10
};
```

Constants
`kFirstMinorNumber`

kInetInterfaceInfoVersion

```
enum {
    kInetInterfaceInfoVersion = 3
};
```

Constants
`kInetInterfaceInfoVersion`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kIP_OPTIONS

```
enum {
    kIP_OPTIONS = 0x01,
    kIP_TOS = 0x02,
    kIP_TTL = 0x03,
    kIP_REUSEADDR = 0x04,
    kIP_DONTROUTE = 0x10,
    kIP_BROADCAST = 0x20,
    kIP_REUSEPORT = 0x0200,
    kIP_HDRINCL = 0x1002,
    kIP_RCVOPTS = 0x1005,
    kIP_RCVDSTADDR = 0x1007,
    kIP_MULTICAST_IF = 0x1010,
    kIP_MULTICAST_TTL = 0x1011,
    kIP_MULTICAST_LOOP = 0x1012,
    kIP_ADD_MEMBERSHIP = 0x1013,
    kIP_DROP_MEMBERSHIP = 0x1014,
    kIP_BROADCAST_IFNAME = 0x1015,
    kIP_RCVIFADDR = 0x1016
};
```

Constants
`kIP_OPTIONS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIP_TOS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIP_TTL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIP_REUSEADDR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIP_DONTROUTE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIP_BROADCAST`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIP_REUSEPORT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIP_HDRINCL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIP_RCVOPTS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIP_RCVDSTADDR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIP_MULTICAST_IF`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIP_MULTICAST_TTL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIP_MULTICAST_LOOP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIP_ADD_MEMBERSHIP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIP_DROP_MEMBERSHIP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIP_BROADCAST_IFNAME`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIP_RCVIFADDR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kIPCPTCPHdrCompressionDisabled

```
enum {
    kIPCPTCPHdrCompressionDisabled = 0,
    kIPCPTCPHdrCompressionEnabled = 1
};
```

Constants

`kIPCPTCPHdrCompressionDisabled`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIPCPTCPHdrCompressionEnabled`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kISDNModuleID

```
enum {
    kISDNModuleID = 7300
};
```

Constants

`kISDNModuleID`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kMaxHostAddrs

```
enum {
    kMaxHostAddrs = 10,
    kMaxSysStringLen = 32,
    kMaxHostNameLen = 255
};
```

Constants

`kMaxHostAddrs`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kMaxSysStringLen`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kMaxHostNameLen`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

Port-Related Constants

Provide length and size values for modules, provider names, and slot IDs.

```
enum {
    kMaxModuleNameLength = 31,
    kMaxModuleNameSize = kMaxModuleNameLength + 1,
    kMaxProviderNameLength = kMaxModuleNameLength + 4,
    kMaxProviderNameSize = kMaxProviderNameLength + 1,
    kMaxSlotIDLength = 7,
    kMaxSlotIDSize = kMaxSlotIDLength + 1,
    kMaxResourceInfoLength = 31,
    kMaxResourceInfoSize = 32,
    kMaxPortNameLength = kMaxModuleNameLength + 4,
    kMaxPortNameSize = kMaxPortNameLength + 1
};
```

Constants

`kMaxModuleNameLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kMaxModuleNameSize`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kMaxProviderNameLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kMaxProviderNameSize`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kMaxSlotIDLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kMaxSlotIDSize`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kMaxResourceInfoLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kMaxResourceInfoSize`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kMaxPortNameLength`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kMaxPortNameSize`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

Discussion
These constants provide length and size values for modules, provider names, and slot IDs. These fields all end with a byte for the terminating zero. The constant kMaxProviderNameSize permits a length of 36 bytes: 31 bytes for the name, up to 4 bytes of extra characters (called minor numbers in STREAMS specifications, and currently not used), and a byte for the zero that terminates the string.

kMaxServices

```
enum {
    kMaxServices = 20
};
```

Constants
`kMaxServices`

kMulticastLength

```
enum {
    kMulticastLength = 6,
    k48BitAddrLength = 6,
    k8022DLSAPLength = 2,
    k8022SNAPLength = 5,
    kEnetAddressLength = k48BitAddrLength + k8022DLSAPLength,
    kSNAPSAP = 0x00AA,
    kIPXSAP = 0x00FF,
    kMax8022SAP = 0x00FE,
    k8022GlobalSAP = 0x00FF,
    kMinDIXSAP = 1501,
    kMaxDIXSAP = 0xFFFF
};
```

Constants
`kMulticastLength`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`k48BitAddrLength`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`k8022DLSAPLength`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`k8022SNAPLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kEnetAddressLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kSNAPSAP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kIPXSAP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kMax8022SAP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`k8022GlobalSAP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kMinDIXSAP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kMaxDIXSAP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kNBPMaxNameLength

```
enum {
    kNBPMaxNameLength = 32,
    kNBPMaxTypeLength = 32,
    kNBPMaxZoneLength = 32,
    kNBPSlushLength = 9,
    kNBPMaxEntityLength = (kNBPMaxNameLength + kNBPMaxTypeLength
+ kNBPMaxZoneLength + 3),
    kNBPEntityBufferSize = (kNBPMaxNameLength + kNBPMaxTypeLength
+ kNBPMaxZoneLength + kNBPSlushLength),
    kNBPWildCard = 0x3D,
    kNBPImbeddedWildCard = 0xC5,
    kNBPDefaultZone = 0x2A
};
```

Constants

`kNBPMaxNameLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kNBPMaxTypeLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kNBPMaxZoneLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kNBPSlushLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kNBPMaxEntityLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kNBPEntityBufferSize`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kNBPWildCard`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kNBPImbeddedWildCard`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kNBPDefaultZone`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kNetbufDataIsOTData

```
enum {
    kNetbufDataIsOTData = 0xFFFFFFFE
};
```

Constants

`kNetbufDataIsOTData`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

kO_ASYNC

Constants

kOTAnyInetAddress

```
enum {
    kOTAnyInetAddress = 0
};
```

Constants

`kOTAnyInetAddress`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kOTAutopushMax

```
enum {
    kOTAutopushMax = 8
};
```

Constants

`kOTAutopushMax`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

kOTCFMClass

```
enum {
    kOTCFMClass = 'otan'
};
```

Constants

`kOTCFMClass`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

kOTDefaultConfigurator

```
enum {
    kOTDefaultConfigurator = 0,
    kOTProtocolFamilyConfigurator = 1,
    kOTLinkDriverConfigurator = 2
};
```

Constants

`kOTDefaultConfigurator`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`kOTProtocolFamilyConfigurator`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`kOTLinkDriverConfigurator`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

kOTFLUSHBAND

```
enum {
    kOTFLUSHBAND = 0x40
};
```

Constants
`kOTFLUSHBAND`

Port Framing Capabilities

Describe a port's framing capabilities.

```
enum {
    kOTFramingEthernet = 0x01,
    kOTFramingEthernetIPX = 0x02,
    kOTFraming8023 = 0x04,
    kOTFraming8022 = 0x08
};
```

Constants
`kOTFramingEthernet`
The port can use standard Ethernet framing.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTFramingEthernetIPX`
The port can use IPX Ethernet framing.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTFraming8023`
The port can use 802.3 Ethernet framing.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTFraming8022`
The port can use 802.2 Ethernet framing.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

Discussion
This enumeration contains flags indicating the type of framing capability that a port has. If the port can handle only one type of framing, this field is 0. This field is dependent on the ports device type.

kOTGenericName

```
enum {
    kOTGenericName = 0
};
```

Constants

`kOTGenericName`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

kOTGetDataSymbol

```
enum {
    kOTGetDataSymbol = 0,
    kOTGetCodeSymbol = 1,
    kOTLoadNewCopy = 2,
    kOTLoadACopy = 4,
    kOTFindACopy = 8,
    kOTLibMask = kOTLoadNewCopy | kOTLoadACopy | kOTFindACopy,
    kOTLoadLibResident = 0x20
};
```

Constants

`kOTGetDataSymbol`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`kOTGetCodeSymbol`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`kOTLoadNewCopy`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`kOTLoadACopy`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`kOTFindACopy`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`kOTLibMask`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`kOTLoadLibResident`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

kOTInitialScan

```
enum {
    kOTInitialScan = 0,
    kOTScanAfterSleep = 1
};
```

Constants

`kOTInitialScan`

`kOTScanAfterSleep`

kOTInvalidPortRef

```
enum {
    kOTInvalidPortRef = 0
};
```

Constants

`kOTInvalidPortRef`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

kOTInvalidRef

```
enum {
    kOTInvalidRef = 0,
    kOTInvalidProviderRef = 0,
    kOTInvalidEndpointRef = 0,
    kOTInvalidMapperRef = 0
};
```

Constants
`kOTInvalidRef`
`kOTInvalidProviderRef`
`kOTInvalidEndpointRef`
`kOTInvalidMapperRef`

kOTInvalidStreamRef

```
enum {
    kOTInvalidStreamRef = 0
};
```

Constants
`kOTInvalidStreamRef`

kOTISDNDefaultCommType

```
enum {
    kOTISDNDefaultCommType = kOTISDNDigital64k,
    kOTISDNDefaultFramingType = kOTISDNFramingHDLC,
    kOTISDNDefault56KAdaptation = kOTISDNNot56KAdaptation
};
```

Constants
`kOTISDNDefaultCommType`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNDefaultFramingType`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNDefault56KAdaptation`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kOTISDNFramingTransparent

```
enum {
    kOTISDNFramingTransparent = 0x0010,
    kOTISDNFramingHDLC = 0x0020,
    kOTISDNFramingV110 = 0x0040,
    kOTISDNFramingV14E = 0x0080
};
```

Constants

`kOTISDNFramingTransparent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNFramingHDLC`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNFramingV110`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNFramingV14E`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kOTISDNFramingTransparentSupported

```
enum {
    kOTISDNFramingTransparentSupported = 0x0010,
    kOTISDNFramingHDLCSupported = 0x0020,
    kOTISDNFramingV110Supported = 0x0040,
    kOTISDNFramingV14ESupported = 0x0080
};
```

Constants

`kOTISDNFramingTransparentSupported`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNFramingHDLCSupported`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNFramingV110Supported`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNFramingV14ESupported`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kOTISDNMaxPhoneSize

```
enum {
    kOTISDNMaxPhoneSize = 32,
    kOTISDNMaxSubSize = 4
};
```

Constants

`kOTISDNMaxPhoneSize`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTISDNMaxSubSize`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

kOTISDNMaxUserDataSize

```
enum {
    kOTISDNMaxUserDataSize = 32
};
```

Constants

`kOTISDNMaxUserDataSize`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

kOTISDNNot56KAdaptation

```
enum {
    kOTISDNNot56KAdaptation = false,
    kOTISDN56KAdaptation = true
};
```

Constants

`kOTISDNNot56KAdaptation`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTISDN56KAdaptation`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

kOTISDNTelephoneALaw

```
enum {
    kOTISDNTelephoneALaw = 1,
    kOTISDNTelephoneMuLaw = 26,
    kOTISDNDigital64k = 13,
    kOTISDNDigital56k = 37,
    kOTISDNVideo64k = 41,
    kOTISDNVideo56k = 42
};
```

Constants

`kOTISDNTelephoneALaw`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNTelephoneMuLaw`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNDigital64k`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNDigital56k`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNVideo64k`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNVideo56k`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kOTISDNUnallocatedNumber

```
enum {
    kOTISDNUnallocatedNumber = 1,
    kOTISDNNoRouteToSpecifiedTransitNetwork = 2,
    kOTISDNNoRouteToDestination = 3,
    kOTISDNChannelUnacceptable = 6,
    kOTISDNNormal = 16,
    kOTISDNUserBusy = 17,
    kOTISDNNoUserResponding = 18,
    kOTISDNNoAnswerFromUser = 19,
    kOTISDNCallRejected = 21,
    kOTISDNNumberChanged = 22,
    kOTISDNNonSelectedUserClearing = 26,
    kOTISDNDestinationOutOfOrder = 27,
    kOTISDNInvalidNumberFormat = 28,
    kOTISDNFacilityRejected = 29,
    kOTISDNNormalUnspecified = 31,
    kOTISDNNoCircuitChannelAvailable = 34,
    kOTISDNNetworkOutOfOrder = 41,
    kOTISDNSwitchingEquipmentCongestion = 42,
    kOTISDNAccessInformationDiscarded = 43,
    kOTISDNRequestedCircuitChannelNotAvailable = 44,
    kOTISDNResourceUnavailableUnspecified = 45,
    kOTISDNQualityOfServiceUnvailable = 49,
    kOTISDNRequestedFacilityNotSubscribed = 50,
    kOTISDNBearerCapabilityNotAuthorized = 57,
    kOTISDNBearerCapabilityNotPresentlyAvailable = 58,
    kOTISDNCallRestricted = 59,
    kOTISDNServiceOrOptionNotAvilableUnspecified = 63,
    kOTISDNBearerCapabilityNotImplemented = 65,
    kOTISDNRequestedFacilityNotImplemented = 69,
    kOTISDNOnlyRestrictedDigitalBearer = 70,
    kOTISDNServiceOrOptionNotImplementedUnspecified = 79,
    kOTISDNCallIdentityNotUsed = 83,
    kOTISDNCallIdentityInUse = 84,
    kOTISDNNoCallSuspended = 85,
    kOTISDNCallIdentityCleared = 86,
    kOTISDNIncompatibleDestination = 88,
    kOTISDNInvalidTransitNetworkSelection = 91,
    kOTISDNInvalidMessageUnspecified = 95,
    kOTISDNMandatoryInformationElementIsMissing = 96,
    kOTISDNMessageTypeNonExistentOrNotImplemented = 97,
    kOTISDNInterworkingUnspecified = 127
};
```

Constants

`kOTISDNUnallocatedNumber`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTISDNNoRouteToSpecifiedTransitNetwork`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTISDNNoRouteToDestination`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNChannelUnacceptable`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNNormal`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNUserBusy`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNNoUserResponding`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNNoAnswerFromUser`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNCallRejected`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNNumberChanged`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNNonSelectedUserClearing`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNDestinationOutOfOrder`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNInvalidNumberFormat`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNFacilityRejected`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNNormalUnspecified`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNNoCircuitChannelAvailable`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNNetworkOutOfOrder`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNSwitchingEquipmentCongestion`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNAccessInformationDiscarded`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNRequestedCircuitChannelNotAvailable`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNResourceUnavailableUnspecified`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNQualityOfServiceUnvailable`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNRequestedFacilityNotSubscribed`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNBearerCapabilityNotAuthorized`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNBearerCapabilityNotPresentlyAvailable`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNCallRestricted`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNServiceOrOptionNotAvilableUnspecified`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNBearerCapabilityNotImplemented`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNRequestedFacilityNotImplemented`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNOnlyRestrictedDigitalBearer`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTISDNServiceOrOptionNotImplementedUnspecified`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTISDNCallIdentityNotUsed`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTISDNCallIdentityInUse`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTISDNNoCallSuspended`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTISDNCallIdentityCleared`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTISDNIncompatibleDestination`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTISDNInvalidTransitNetworkSelection`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTISDNInvalidMessageUnspecified`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTISDNMandatoryInformationElementIsMissing`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTISDNMessageTypeNonExistentOrNotImplemented`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTISDNInterworkingUnspecified`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

kOTLastSlotNumber

```
enum {
    kOTLastSlotNumber = 255,
    kOTLastOtherNumber = 255
};
```

Constants

`kOTLastSlotNumber`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTLastOtherNumber`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

kOTLvlFatal

```
enum {
    kOTLvlFatal = 0,
    kOTLvlNonfatal = 1,
    kOTLvlExtFatal = 2,
    kOTLvlExtNonfatal = 3,
    kOTLvlUserErr = 4,
    kOTLvlInfoErr = 5,
    kOTLvlInfoOnly = 6
};
```

Constants
kOTLvlFatal
kOTLvlNonfatal
kOTLvlExtFatal
kOTLvlExtNonfatal
kOTLvlUserErr
kOTLvlInfoErr
kOTLvlInfoOnly

kOTMinimumTimerValue

```
enum {
    kOTMinimumTimerValue = 8
};
```

Constants
kOTMinimumTimerValue

kOTModIsDriver

```
enum {
    kOTModIsDriver = 1,
    kOTModIsModule = 2,
    kOTModNoWriter = 16,
    kOTModUpperIsTPI = 4096,
    kOTModUpperIsDLPI = 8192,
    kOTModLowerIsTPI = 16384,
    kOTModLowerIsDLPI = 32768,
    kOTModGlobalContext = 8388608,
    kOTModUsesInterrupts = 134217728,
    kOTModIsComplexDriver = 536870912,
    kOTModIsFilter = 1073741824
};
```

Constants
kOTModIsDriver
kOTModIsModule
kOTModNoWriter
kOTModUpperIsTPI
kOTModUpperIsDLPI
kOTModLowerIsTPI

```
kOTModLowerIsDLPI
kOTModGlobalContext
kOTModUsesInterrupts
kOTModIsComplexDriver
kOTModIsFilter
```

kOTNetbufDataIsOTBufferStar

```
enum {
    kOTNetbufDataIsOTBufferStar = 0xFFFFFFFD
};
```

Constants

`kOTNetbufDataIsOTBufferStar`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

kOTNetbufIsRawMode

```
enum {
    kOTNetbufIsRawMode = 0xFFFFFFFF
};
```

Constants

`kOTNetbufIsRawMode`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

kOTNoMemoryConfigurationPtr

```
enum {
    kOTNoMemoryConfigurationPtr = 0,
    kOTInvalidConfigurationPtr = -1
};
```

Constants

`kOTNoMemoryConfigurationPtr`

`kOTInvalidConfigurationPtr`

kOTNoMessagesAvailable

```
enum {
    kOTNoMessagesAvailable = 0xFFFFFFFF,
    kOTAnyMsgType = 0xFFFFFFFE,
    kOTDataMsgTypes = 0xFFFFFFFC,
    kOTMProtoMsgTypes = 0xFFFFFFFB,
    kOTOnlyMProtoMsgTypes = 0xFFFFFFFA
};
```

Constants

`kOTNoMessagesAvailable`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`kOTAnyMsgType`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`kOTDataMsgTypes`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`kOTMProtoMsgTypes`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`kOTOnlyMProtoMsgTypes`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

kOTOptionHeaderSize

```
enum {
    kOTOptionHeaderSize = sizeof(TOptionHeader),
    kOTBooleanOptionDataSize = sizeof(UInt32),
    kOTBooleanOptionSize = kOTOptionHeaderSize + kOTBooleanOptionDataSize,
    kOTOneByteOptionSize = kOTOptionHeaderSize + 1,
    kOTTwoByteOptionSize = kOTOptionHeaderSize + 2,
    kOTFourByteOptionSize = kOTOptionHeaderSize + sizeof(UInt32)
};
```

Constants

`kOTOptionHeaderSize`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTBooleanOptionDataSize`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTBooleanOptionSize`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTOneByteOptionSize`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTTwoByteOptionSize`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTFourByteOptionSize`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

kOTPCINoErrorStayLoaded

```
enum {
    kOTPCINoErrorStayLoaded = 1
};
```

Constants

`kOTPCINoErrorStayLoaded`

Port Flags

Specify a port's status.

```
enum {
    kOTPortIsActive = 0x00000001,
    kOTPortIsDisabled = 0x00000002,
    kOTPortIsUnavailable = 0x00000004,
    kOTPortIsOffline = 0x00000008
};
```

Constants

`kOTPortIsActive`
The port is in use.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPortIsDisabled`
The port may or may not be in use, but no other client can use it.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPortIsUnavailable`
The port is not available for use.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPortIsOffline`
The port is off-line. This bit is typically only set when the port is active, the port autoconnects, and it is currently not connected.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

Port Additional Flags

Specify additional information about a port.

```
enum {
    kOTPortIsDLPI = 0x00000001,
    kOTPortIsTPI = 0x00000002,
    kOTPortCanYield = 0x00000004,
    kOTPortCanArbitrate = 0x00000008,
    kOTPortIsTransitory = 0x00000010,
    kOTPortAutoConnects = 0x00000020,
    kOTPortIsSystemRegistered = 0x00004000,
    kOTPortIsPrivate = 0x00008000,
    kOTPortIsAlias = 0x80000000
};
```

Constants

`kOTPortIsDLPI`
The port driver is a DLPI STREAMS module.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPortIsTPI`
The port driver is a TPI STREAMS module.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPortCanYield`
The port can yield when requested.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPortCanArbitrate`
Reserved.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPortIsTransitory`
The port has off-line/on-line status.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPortAutoConnects`
The port auto connects. The port goes on-line and off-line on demand. ISDN is a typical example.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPortIsSystemRegistered`
The port was registered by the system from the Name Registry
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPortIsPrivate`
The port is private.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPortIsAlias`
The port is an alias for another port.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

kOTPrintOnly

```
enum {
    kOTPrintOnly = 0,
    kOTPrintThenStop = 1
};
```

Constants

`kOTPrintOnly`

`kOTPrintThenStop`

kOTRawRcvOn

```
enum {
    kOTRawRcvOn = 0,
    kOTRawRcvOff = 1,
    kOTRawRcvOnWithTimeStamp = 2
};
```

Constants

`kOTRawRcvOn`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTRawRcvOff`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTRawRcvOnWithTimeStamp`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

kOTSerialDefaultBaudRate

```
enum {
    kOTSerialDefaultBaudRate = 19200,
    kOTSerialDefaultDataBits = 8,
    kOTSerialDefaultStopBits = 10,
    kOTSerialDefaultParity = kOTSerialNoParity,
    kOTSerialDefaultHandshake = 0,
    kOTSerialDefaultOnChar = ('Q' & 0xFFFFFFBF),
    kOTSerialDefaultOffChar = ('S' & 0xFFFFFFBF),
    kOTSerialDefaultSndBufSize = 1024,
    kOTSerialDefaultRcvBufSize = 1024,
    kOTSerialDefaultSndLoWat = 96,
    kOTSerialDefaultRcvLoWat = 1,
    kOTSerialDefaultRcvTimeout = 10
};
```

Constants

`kOTSerialDefaultBaudRate`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialDefaultDataBits`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialDefaultStopBits`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialDefaultParity`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialDefaultHandshake`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialDefaultOnChar`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialDefaultOffChar`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialDefaultSndBufSize`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialDefaultRcvBufSize`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialDefaultSndLoWat`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialDefaultRcvLoWat`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialDefaultRcvTimeout`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kOTSerialFramingAsync

```
enum {
    kOTSerialFramingAsync = 0x01,
    kOTSerialFramingHDLC = 0x02,
    kOTSerialFramingSDLC = 0x04,
    kOTSerialFramingAsyncPackets = 0x08,
    kOTSerialFramingPPP = 0x10
};
```

Constants

`kOTSerialFramingAsync`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialFramingHDLC`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialFramingSDLC`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialFramingAsyncPackets`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialFramingPPP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kOTSerialSwOverRunErr

```
enum {
    kOTSerialSwOverRunErr = 0x01,
    kOTSerialBreakOn = 0x08,
    kOTSerialParityErr = 0x10,
    kOTSerialOverrunErr = 0x20,
    kOTSerialFramingErr = 0x40,
    kOTSerialXOffSent = 0x00010000,
    kOTSerialDTRNegated = 0x00020000,
    kOTSerialCTLHold = 0x00040000,
    kOTSerialXOffHold = 0x00080000,
    kOTSerialOutputBreakOn = 0x01000000
};
```

Constants

`kOTSerialSwOverRunErr`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialBreakOn`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialParityErr`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialOverrunErr`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialFramingErr`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialXOffSent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialDTRNegated`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialCTLHold`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialXOffHold`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialOutputBreakOn`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kOTSerialXOnOffInputHandshake

```
enum {
    kOTSerialXOnOffInputHandshake = 1,
    kOTSerialXOnOffOutputHandshake = 2,
    kOTSerialCTSInputHandshake = 4,
    kOTSerialDTROutputHandshake = 8
};
```

Constants

`kOTSerialXOnOffInputHandshake`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialXOnOffOutputHandshake`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialCTSInputHandshake`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kOTSerialDTROutputHandshake`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kOTSpecificConfigPass

```
enum {
    kOTSpecificConfigPass = 0,
    kOTGenericConfigPass = 1
};
```

Constants

`kOTSpecificConfigPass`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`kOTGenericConfigPass`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

kOTT_BIND_REQ

```
enum {
    kOTT_BIND_REQ = 101,
    kOTT_CONN_REQ = 102,
    kOTT_CONN_RES = 103,
    kOTT_DATA_REQ = 104,
    kOTT_DISCON_REQ = 105,
    kOTT_EXDATA_REQ = 106,
    kOTT_INFO_REQ = 107,
    kOTT_OPTMGMT_REQ = 108,
    kOTT_ORDREL_REQ = 109,
    kOTT_UNBIND_REQ = 110,
    kOTT_UNITDATA_REQ = 111,
    kOTT_ADDR_REQ = 112,
    kOTT_UREQUEST_REQ = 113,
    kOTT_REQUEST_REQ = 114,
    kOTT_UREPLY_REQ = 115,
    kOTT_REPLY_REQ = 116,
    kOTT_CANCELREQUEST_REQ = 117,
    kOTT_CANCELREPLY_REQ = 118,
    kOTT_REGNAME_REQ = 119,
    kOTT_DELNAME_REQ = 120,
    kOTT_LKUPNAME_REQ = 121,
    kOTT_BIND_ACK = 122,
    kOTT_CONN_CON = 123,
    kOTT_CONN_IND = 124,
    kOTT_DATA_IND = 125,
    kOTT_DISCON_IND = 126,
    kOTT_ERROR_ACK = 127,
    kOTT_EXDATA_IND = 128,
    kOTT_INFO_ACK = 129,
    kOTT_OK_ACK = 130,
    kOTT_OPTMGMT_ACK = 131,
    kOTT_ORDREL_IND = 132,
    kOTT_UNITDATA_IND = 133,
    kOTT_UDERROR_IND = 134,
    kOTT_ADDR_ACK = 135,
    kOTT_UREQUEST_IND = 136,
    kOTT_REQUEST_IND = 137,
    kOTT_UREPLY_IND = 138,
    kOTT_REPLY_IND = 139,
    kOTT_UREPLY_ACK = 140,
    kOTT_REPLY_ACK = 141,
    kOTT_RESOLVEADDR_REQ = 142,
    kOTT_RESOLVEADDR_ACK = 143,
    kOTT_LKUPNAME_CON = 146,
    kOTT_LKUPNAME_RES = 147,
    kOTT_REGNAME_ACK = 148,
    kOTT_SEQUENCED_ACK = 149,
    kOTT_EVENT_IND = 160
};
```

Constants

kOTT_BIND_REQ
kOTT_CONN_REQ
kOTT_CONN_RES
kOTT_DATA_REQ

```
kOTT_DISCON_REQ
kOTT_EXDATA_REQ
kOTT_INFO_REQ
kOTT_OPTMGMT_REQ
kOTT_ORDREL_REQ
kOTT_UNBIND_REQ
kOTT_UNITDATA_REQ
kOTT_ADDR_REQ
kOTT_UREQUEST_REQ
kOTT_REQUEST_REQ
kOTT_UREPLY_REQ
kOTT_REPLY_REQ
kOTT_CANCELREQUEST_REQ
kOTT_CANCELREPLY_REQ
kOTT_REGNAME_REQ
kOTT_DELNAME_REQ
kOTT_LKUPNAME_REQ
kOTT_BIND_ACK
kOTT_CONN_CON
kOTT_CONN_IND
kOTT_DATA_IND
kOTT_DISCON_IND
kOTT_ERROR_ACK
kOTT_EXDATA_IND
kOTT_INFO_ACK
kOTT_OK_ACK
kOTT_OPTMGMT_ACK
kOTT_ORDREL_IND
kOTT_UNITDATA_IND
kOTT_UDERROR_IND
kOTT_ADDR_ACK
kOTT_UREQUEST_IND
kOTT_REQUEST_IND
kOTT_UREPLY_IND
kOTT_REPLY_IND
kOTT_UREPLY_ACK
kOTT_REPLY_ACK
kOTT_RESOLVEADDR_REQ
kOTT_RESOLVEADDR_ACK
kOTT_LKUPNAME_CON
kOTT_LKUPNAME_RES
kOTT_REGNAME_ACK
kOTT_SEQUENCED_ACK
kOTT_EVENT_IND
```

kOTT_TIMER_REQ

```
enum {
    kOTT_TIMER_REQ = 80,
    kOTT_MIB_REQ = 81,
    kOTT_MIB_ACK = 82,
    kOTT_PRIVATE_REQ = 90
```

```
};
```

Constants
kOTT_TIMER_REQ
kOTT_MIB_REQ
kOTT_MIB_ACK
kOTT_PRIVATE_REQ

kOTTRANSPARENT

```
enum {
    kOTTRANSPARENT = 0xFFFFFFFF
};
```

Constants
kOTTRANSPARENT

kPPPAsyncMapCharsNone

```
enum {
    kPPPAsyncMapCharsNone = 0x00000000,
    kPPPAsyncMapCharsXOnXOff = 0x000A0000,
    kPPPAsyncMapCharsAll = 0xFFFFFFFF
};
```

Constants
kPPPAsyncMapCharsNone
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProviders.h.
kPPPAsyncMapCharsXOnXOff
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProviders.h.
kPPPAsyncMapCharsAll
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProviders.h.

kPPPCompressionDisabled

```
enum {
    kPPPCompressionDisabled = 0x00000000,
    kPPPProtoCompression = 0x00000001,
    kPPPAddrCompression = 0x00000002
};
```

Constants
kPPPCompressionDisabled
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProviders.h.

`kPPPProtoCompression`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPAddrCompression`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kPPPConnectionStatusDialogsFlag

```
enum {
    kPPPConnectionStatusDialogsFlag = 0x00000001,
    kPPPConnectionRemindersFlag = 0x00000002,
    kPPPConnectionFlashingIconFlag = 0x00000004,
    kPPPOutPasswordDialogsFlag = 0x00000008,
    kPPPAllAlertsDisabledFlag = 0x00000000,
    kPPPAllAlertsEnabledFlag = 0x0000000F
};
```

Constants

`kPPPConnectionStatusDialogsFlag`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPConnectionRemindersFlag`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPConnectionFlashingIconFlag`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPOutPasswordDialogsFlag`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPAllAlertsDisabledFlag`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPAllAlertsEnabledFlag`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kPPPConnectionStatusIdle

```
enum {
    kPPPConnectionStatusIdle = 1,
    kPPPConnectionStatusConnecting = 2,
    kPPPConnectionStatusConnected = 3,
    kPPPConnectionStatusDisconnecting = 4
};
```

Constants

`kPPPConnectionStatusIdle`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPConnectionStatusConnecting`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPConnectionStatusConnected`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPConnectionStatusDisconnecting`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kPPPEvent

```
enum {
    kPPPEvent = kPROTOCOLEVENT | 0x000F0000,
    kPPPConnectCompleteEvent = kPPPEvent + 1,
    kPPPSetScriptCompleteEvent = kPPPEvent + 2,
    kPPPDisconnectCompleteEvent = kPPPEvent + 3,
    kPPPDisconnectEvent = kPPPEvent + 4,
    kPPPIPCPUpEvent = kPPPEvent + 5,
    kPPPIPCPDownEvent = kPPPEvent + 6,
    kPPPLCPUpEvent = kPPPEvent + 7,
    kPPPLCPDownEvent = kPPPEvent + 8,
    kPPPLowerLayerUpEvent = kPPPEvent + 9,
    kPPPLowerLayerDownEvent = kPPPEvent + 10,
    kPPPAuthenticationStartedEvent = kPPPEvent + 11,
    kPPPAuthenticationFinishedEvent = kPPPEvent + 12,
    kPPPDCEInitStartedEvent = kPPPEvent + 13,
    kPPPDCEInitFinishedEvent = kPPPEvent + 14,
    kPPPDCECallStartedEvent = kPPPEvent + 15,
    kPPPDCECallFinishedEvent = kPPPEvent + 16
};
```

Constants

`kPPPEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPConnectCompleteEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPSetScriptCompleteEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPDisconnectCompleteEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPDisconnectEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPIPCPUpEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPIPCPDownEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPLCPUpEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPLCPDownEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPLowerLayerUpEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPLowerLayerDownEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPAuthenticationStartedEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPAuthenticationFinishedEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPDCEInitStartedEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPDCEInitFinishedEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPDCECallStartedEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPDCECallFinishedEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kPPPMaxIDLength

```
enum {
    kPPPMaxIDLength = 255,
    kPPPMaxPasswordLength = 255,
    kPPPMaxDTEAddressLength = 127,
    kPPPMaxCallInfoLength = 255
};
```

Constants

`kPPPMaxIDLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPMaxPasswordLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPMaxDTEAddressLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPMaxCallInfoLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kPPPMinMRU

```
enum {
    kPPPMinMRU = 0,
    kPPPMaxMRU = 4500
};
```

Constants

`kPPPMinMRU`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPMaxMRU`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kPPPNoOutAuthentication

```
enum {
    kPPPNoOutAuthentication = 0,
    kPPPCHAPOrPAPOutAuthentication = 1
};
```

Constants

`kPPPNoOutAuthentication`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPCHAPOrPAPOutAuthentication`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kPPPScriptTypeModem

```
enum {
    kPPPScriptTypeModem = 1,
    kPPPScriptTypeConnect = 2,
    kPPPMaxScriptSize = 32000
};
```

Constants

`kPPPScriptTypeModem`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPScriptTypeConnect`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPMaxScriptSize`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kPPPStateInitial

```
enum {
    kPPPStateInitial = 1,
    kPPPStateClosed = 2,
    kPPPStateClosing = 3,
    kPPPStateOpening = 4,
    kPPPStateOpened = 5
};
```

Constants

`kPPPStateInitial`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPStateClosed`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPStateClosing`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPStateOpening`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kPPPStateOpened`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kRAProductClientOnly

```
enum {
    kRAProductClientOnly = 2,
    kRAProductOnePortServer = 3,
    kRAProductManyPortServer = 4
};
```

Constants

`kRAProductClientOnly`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kRAProductOnePortServer`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kRAProductManyPortServer`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kSAP_ONE

```
enum {
    kSAP_ONE = 1,
    kSAP_RANGE = 2,
    kSAP_ALL = 3,
    kSAP_CLEAR = 4
};
```

Constants

`kSAP_ONE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`kSAP_RANGE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`kSAP_ALL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`kSAP_CLEAR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

kSerialABModuleID

```
enum {
    kSerialABModuleID = 7200
};
```

Constants

`kSerialABModuleID`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kSIGHUP

```
enum {
    kSIGHUP = 1,
    kSIGURG = 16,
    kSIGPOLL = 30
};
```

Constants

`kSIGHUP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kSIGURG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kSIGPOLL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

kT_UNSPEC

```
enum {
    kT_UNSPEC = 0xFFFFFFFD,
    T_ALLOPT = 0
};
```

Constants

`kT_UNSPEC`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`T_ALLOPT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

kT8022HeaderLength

```
enum {
    kT8022HeaderLength = 3,
    kT8022SNAPHeaderLength = 3 + k8022SNAPLength,
   kT8022FullPacketHeaderLength = kEnetPacketHeaderLength + kT8022SNAPHeaderLength
};
```

Constants

`kT8022HeaderLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kT8022SNAPHeaderLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kT8022FullPacketHeaderLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kT8022ModuleID

```
enum {
    kT8022ModuleID = 7100,
    kEnetModuleID = 7101,
    kTokenRingModuleID = 7102,
    kFDDIModuleID = 7103
};
```

Constants

`kT8022ModuleID`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kEnetModuleID`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kTokenRingModuleID`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kFDDIModuleID`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

kZIPMaxZoneLength

```
enum {
    kZIPMaxZoneLength = kNBPMaxZoneLength
};
```

Constants
`kZIPMaxZoneLength`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

LNK_ENET

```
enum {
    LNK_ENET = 'ENET',
    LNK_TOKN = 'TOKN',
    LNK_FDDI = 'FDDI',
    LNK_TPI = 'LTPI'
};
```

Constants
`LNK_ENET`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`LNK_TOKN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`LNK_FDDI`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`LNK_TPI`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

LOGMSGSZ

```
enum {
    LOGMSGSZ = 128
};
```

Constants

`LOGMSGSZ`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

M_MI

```
enum {
    M_MI = 0x40,
    M_MI_READ_RESET = 1,
    M_MI_READ_SEEK = 2,
    M_MI_READ_END = 4
};
```

Constants

`M_MI`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_MI_READ_RESET`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_MI_READ_SEEK`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_MI_READ_END`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

MIOC_ISDN

```
enum {
    MIOC_ISDN = 85
};
```

Constants

`MIOC_ISDN`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

MIOC_STREAMIO

```
enum {
    MIOC_STREAMIO = 65,
    MIOC_TMOD = 'a',
    MIOC_STRLOG = 'b',
    MIOC_ND = 'c',
    MIOC_ECHO = 'd',
    MIOC_TLI = 'e',
    MIOC_RESERVEDf = 'f',
    MIOC_SAD = 'g',
    MIOC_ARP = 'h',
    MIOC_HAVOC = 72,
    MIOC_RESERVEDi = 'i',
    MIOC_SIOC = 'j',
    MIOC_TCP = 'k',
    MIOC_DLPI = 'l',
    MIOC_SOCKETS = 'm',
    MIOC_IPX = 'o',
    MIOC_OT = 79,
    MIOC_ATALK = 84,
    MIOC_SRL = 85,
    MIOC_RESERVEDp = 'p',
    MIOC_RESERVEDr = 'r',
    MIOC_RESERVEDs = 's',
    MIOC_CFIG = 'z'
};
```

Constants

`MIOC_STREAMIO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_TMOD`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_STRLOG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_ND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_ECHO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_TLI`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_RESERVEDf`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_SAD`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_ARP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_HAVOC`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_RESERVEDi`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_SIOC`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_TCP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_DLPI`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_SOCKETS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_IPX`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_OT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_ATALK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_SRL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_RESERVEDp`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_RESERVEDr`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_RESERVEDs`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`MIOC_CFIG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

MORECTL

```
enum {
    MORECTL = 0x01,
    MOREDATA = 0x02
};
```

Constants

`MORECTL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`MOREDATA`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

MSG_HIPRI

```
enum {
    MSG_HIPRI = 0x01,
    MSG_BAND = 0x02,
    MSG_ANY = 0x04
};
```

Constants

`MSG_HIPRI`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`MSG_BAND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`MSG_ANY`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

MSGMARK

```
enum {
    MSGMARK = 0x01,
    MSGNOLOOP = 0x02,
    MSGDELIM = 0x04,
    MSGNOGET = 0x08
};
```

Constants

`MSGMARK`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`MSGNOLOOP`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`MSGDELIM`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`MSGNOGET`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

MUXID_ALL

```
enum {
    MUXID_ALL = -1
};
```

Constants

`MUXID_ALL`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

NOERROR

```
enum {
    NOERROR = -1
};
```

Constants

`NOERROR`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

O_ASYNC

```
enum {
    O_ASYNC = kO_ASYNC,
    O_NDELAY = kO_NDELAY,
    O_NONBLOCK = kO_NONBLOCK
};
```

Constants

`O_ASYNC`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`O_NDELAY`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`O_NONBLOCK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

OPT_ADDMCAST

```
enum {
    OPT_ADDMCAST = 0x1000,
    OPT_DELMCAST = 0x1001,
    OPT_RCVPACKETTYPE = 0x1002,
    OPT_RCVDESTADDR = 0x1003,
    OPT_SETRAWMODE = 0x1004,
    OPT_SETPROMISCUOUS = 0x1005
};
```

Constants

`OPT_ADDMCAST`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`OPT_DELMCAST`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`OPT_RCVPACKETTYPE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`OPT_RCVDESTADDR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`OPT_SETRAWMODE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`OPT_SETPROMISCUOUS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

Bus Type Constants

Specify the bus type of a port.

```
typedef UInt8 OTBusType;
enum {
    kOTUnknownBusPort = 0,
    kOTMotherboardBus = 1,
    kOTNuBus = 2,
    kOTPCIBus = 3,
    kOTGeoPort = 4,
    kOTPCCardBus = 5,
    kOTFireWireBus = 6,
    kOTLastBusIndex = 15
};
```

Constants

`kOTUnknownBusPort`
The port's bus type is not a known type.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTMotherboardBus`
The port is on the motherboard.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTNuBus`
The port is on a NuBus.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPCIBus`
The port is on a PCI bus.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTGeoPort`
The port is a GeoPort device.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPCCardBus`
The port is on a PCCard bus.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTFireWireBus`
The port is on a Firewire bus.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTLastBusIndex`
The maximum bus type that the port can support.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

Hardware Device Types

Lists hardware device types for Open Transport ports.

```
typedef UInt16 OTDeviceType;
enum {
    kOTNoDeviceType = 0,
    kOTADEVDevice = 1,
    kOTMDEVDevice = 2,
    kOTLocalTalkDevice = 3,
    kOTIRTalkDevice = 4,
    kOTTokenRingDevice = 5,
    kOTISDNDevice = 6,
    kOTATMDevice = 7,
    kOTSMDSDevice = 8,
    kOTSerialDevice = 9,
    kOTEthernetDevice = 10,
    kOTSLIPDevice = 11,
    kOTPPPDevice = 12,
    kOTModemDevice = 13,
    kOTFastEthernetDevice = 14,
    kOTFDDIDevice = 15,
    kOTIrDADevice = 16,
    kOTATMSNAPDevice = 17,
    kOTFibreChannelDevice = 18,
    kOTFireWireDevice = 19,
    kOTPseudoDevice = 1023,
    kOTLastDeviceIndex = 1022
};
```

Constants

`kOTNoDeviceType`
The port's device type is not specified. This value is illegal.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTADEVDevice`
The port is specified as an 'adev' device, which is a pseudodevice used by AppleTalk.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTMDEVDevice`
The port is specified as an 'mdev' device, which is a pseudodevice used by TCP.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTLocalTalkDevice`
The port is specified as a LocalTalk device.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTIRTalkDevice`
The port is specified as an IRTalk device.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTTokenRingDevice`
The port is specified as a token ring device.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTISDNDevice`
The port is specified as an ISDN device.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTATMDevice`
The port is specified as an ATM device.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTSMDSDevice`
The port is specified as a SMDS device.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTSerialDevice`
The port is specified as a serial device.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTEthernetDevice`
The port is specified as an Ethernet device.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTSLIPDevice`
The port is specified as a SLIP pseudodevice.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPPPDevice`
The port is specified as a SLIP pseudodevice.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTModemDevice`
The port is specified as a modem pseudodevice.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTFastEthernetDevice`
The port is specified as an 100 MB Ethernet device.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTFDDIDevice`
The port is specified as a FDDI device.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTIrDADevice`
The port is specified as an IrDA Infrared device.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTATMSNAPDevice`
The port is specified as an ATM pseudodevice simulating a SNAP device.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTFibreChannelDevice`
The port is specified as a Fibre Channel device.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTFireWireDevice`
The port is specified as a Firewire device.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPseudoDevice`
The port is designated as a pseudodevice.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTLastDeviceIndex`
The maximum device types that a port can use.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

Special Considerations

Do not arbitrarily add new device types. Please contact Developer Support at Apple Computer, Inc. to obtain a new, unique device type.

OTInitializationFlags

```
typedef UInt32 OTInitializationFlags;
enum {
    kInitOTForApplicationMask = 1,
    kInitOTForExtensionMask = 2
};
```

Constants

`kInitOTForApplicationMask`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kInitOTForExtensionMask`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

OTOpenFlags

```
typedef UInt32 OTOpenFlags;
enum {
    kO_ASYNC = 0x01,
    kO_NDELAY = 0x04,
    kO_NONBLOCK = 0x04
};
```

Constants

`kO_ASYNC`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kO_NDELAY`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kO_NONBLOCK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

OTPacketType

```
typedef UInt32 OTPacketType;
enum {
    kETypeStandard = 0,
    kETypeMulticast = 1,
    kETypeBroadcast = 2,
    kETRawPacketBit = 0x80000000,
    kETTimeStampBit = 0x40000000
};
```

Constants

`kETypeStandard`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kETypeMulticast`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kETypeBroadcast`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kETRawPacketBit`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`kETTimeStampBit`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

Endpoint Service Types

Contains values that Open Transport can return in the `servtype` field of the `TEndpointInfo` structure.

```
typedef UInt32 OTServiceType;
enum {
    T_COTS = 1,
    T_COTS_ORD = 2,
    T_CLTS = 3,
    T_TRANS = 5,
    T_TRANS_ORD = 6,
    T_TRANS_CLTS = 7
};
```

Constants

`T_COTS`
Connection-oriented transactionless service without orderly release.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`T_COTS_ORD`
Connection-oriented transactionless service with optional orderly release.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_CLTS`
Connectionless transactionless service.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_TRANS`
Connection-oriented transaction-based service without orderly release.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_TRANS_ORD`
Connection-oriented transaction-based service with optional orderly release.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_TRANS_CLTS`
Connectionless transaction-based service.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

Endpoint States

Define enpoint states for the `OTGetEndpointState` function.

```
typedef UInt32 OTXTIStates;
enum {
    T_UNINIT = 0,
    T_UNBND = 1,
    T_IDLE = 2,
    T_OUTCON = 3,
    T_INCON = 4,
    T_DATAXFER = 5,
    T_OUTREL = 6,
    T_INREL = 7
};
```

Constants

`T_UNINIT`
This endpoint has been closed and destroyed.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_UNBND`
This endpoint is initialized but has not yet been bound to an address.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_IDLE`
This endpoint has been bound to an address and is ready for use: connectionless endpoints can send or receive data; connection-oriented endpoints can initiate or listen for a connection.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_OUTCON`
This endpoint has initiated a connection and is waiting for the peer endpoint to accept the connection.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_INCON`
This endpoint has received a connection request but has not yet accepted or rejected the request.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_DATAXFER`
This connection-oriented endpoint can now transfer data because the connection has been established.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_OUTREL`
This endpoint has issued an orderly disconnect that the peer has not acknowledged. The endpoint can continue to read data, but must not send any more data.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_INREL`
This endpoint has received a request for an orderly disconnect, which it has not yet acknowledged. The endpoint can continue to send data until it acknowledges the disconnection request, but it must not read data.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

ParityOptionValues

```
typedef UInt32 ParityOptionValues;
enum {
    kOTSerialNoParity = 0,
    kOTSerialOddParity = 1,
    kOTSerialEvenParity = 2
};
```

Constants

`kOTSerialNoParity`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTSerialOddParity`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`kOTSerialEvenParity`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

QB_FULL

```
enum {
    QB_FULL = 0x01,
    QB_WANTW = 0x02,
    QB_BACK = 0x04
};
```

Constants

`QB_FULL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QB_WANTW`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QB_BACK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

qfields

```
typedef SInt32 qfields;
enum {
    QHIWAT = 0,
    QLOWAT = 1,
    QMAXPSZ = 2,
    QMINPSZ = 3,
    QCOUNT = 4,
    QFIRST = 5,
    QLAST = 6,
    QFLAG = 7,
    QBAD = 8
};
```

Constants

`QHIWAT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QLOWAT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QMAXPSZ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QMINPSZ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QCOUNT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QFIRST`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QLAST`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QFLAG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QBAD`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

QNORM

```
enum {
    QNORM = 0,
    M_DATA = 0,
    M_PROTO = 1,
    M_BREAK = 0x08,
    M_PASSFP = 0x09,
    M_SIG = 0x0B,
    M_DELAY = 0x0C,
    M_CTL = 0x0D,
    M_IOCTL = 0x0E,
    M_SETOPTS = 0x10,
    M_RSE = 0x11
};
```

Constants

`QNORM`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`M_DATA`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`M_PROTO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`M_BREAK`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_PASSFP`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_SIG`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_DELAY`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_CTL`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_IOCTL`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_SETOPTS`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_RSE`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

QPCTL

```
enum {
    QPCTL = 0x80,
    M_IOCACK = 0x81,
    M_IOCNAK = 0x82,
    M_PCPROTO = 0x83,
    M_PCSIG = 0x84,
    M_FLUSH = 0x86,
    M_STOP = 0x87,
    M_START = 0x88,
    M_HANGUP = 0x89,
    M_ERROR = 0x8A,
    M_READ = 0x8B,
    M_COPYIN = 0x8C,
    M_COPYOUT = 0x8D,
    M_IOCDATA = 0x8E,
    M_PCRSE = 0x90,
    M_STOPI = 0x91,
    M_STARTI = 0x92,
    M_HPDATA = 0x93
};
```

Constants

`QPCTL`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_IOCACK`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_IOCNAK`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_PCPROTO`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_PCSIG`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_FLUSH`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_STOP`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_START`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`M_HANGUP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`M_ERROR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`M_READ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`M_COPYIN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`M_COPYOUT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`M_IOCDATA`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`M_PCRSE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`M_STOPI`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`M_STARTI`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`M_HPDATA`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

QREADR

```
enum {
    QREADR = 0x01,
    QNOENB = 0x02,
    QFULL = 0x04,
    QWANTR = 0x08,
    QWANTW = 0x10,
    QUSE = 0x20,
    QENAB = 0x40,
    QBACK = 0x80,
    QOLD = 0x0100,
    QHLIST = 0x0200,
    QWELDED = 0x0400,
    QUNWELDING = 0x0800,
    QPROTECTED = 0x1000,
    QEXCOPENCLOSE = 0x2000
};
```

Constants

`QREADR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QNOENB`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QFULL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QWANTR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QWANTW`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QUSE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QENAB`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QBACK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QOLD`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QHLIST`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QWELDED`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QUNWELDING`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QPROTECTED`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`QEXCOPENCLOSE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

RNORM

```
enum {
    RNORM = 0x01,
    RMSGD = 0x02,
    RMSGN = 0x04,
    RFILL = 0x08
};
```

Constants

`RNORM`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`RMSGD`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`RMSGN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`RFILL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

RPROTNORM

```
enum {
    RPROTNORM = 0x10,
    RPROTDIS = 0x20,
    RPROTDAT = 0x40
};
```

Constants

`RPROTNORM`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`RPROTDIS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`RPROTDAT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

RS_EXDATA

```
enum {
    RS_EXDATA = 0x20,
    RS_ALLOWAGAIN = 0x40,
    RS_DELIMITMSG = 0x80
};
```

Constants

`RS_EXDATA`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`RS_ALLOWAGAIN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`RS_DELIMITMSG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

RS_HIPRI

```
enum {
    RS_HIPRI = 0x01
};
```

Constants

`RS_HIPRI`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

S_INPUT

```
enum {
    S_INPUT = 0x01,
    S_HIPRI = 0x02,
    S_OUTPUT = 0x04,
    S_MSG = 0x08,
    S_RDNORM = 0x10,
    S_RDBAND = 0x20,
    S_WRNORM = 0x40,
    S_WRBAND = 0x80,
    S_ERROR = 0x0100,
    S_HANGUP = 0x0200,
    S_BANDURG = 0x0400
};
```

Constants

`S_INPUT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`S_HIPRI`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`S_OUTPUT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`S_MSG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`S_RDNORM`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`S_RDBAND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`S_WRNORM`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`S_WRBAND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`S_ERROR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`S_HANGUP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`S_BANDURG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

SENDZERO

```
enum {
    SENDZERO = 0x0001,
    XPG4_1 = 0x0002
};
```

Constants

`SENDZERO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`XPG4_1`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

SERIAL_OPT_BAUDRATE

```
enum {
    SERIAL_OPT_BAUDRATE = 0x0100,
    SERIAL_OPT_DATABITS = 0x0101,
    SERIAL_OPT_STOPBITS = 0x0102,
    SERIAL_OPT_PARITY = 0x0103,
    SERIAL_OPT_STATUS = 0x0104,
    SERIAL_OPT_HANDSHAKE = 0x0105,
    SERIAL_OPT_RCVTIMEOUT = 0x0106,
    SERIAL_OPT_ERRORCHARACTER = 0x0107,
    SERIAL_OPT_EXTCLOCK = 0x0108,
    SERIAL_OPT_BURSTMODE = 0x0109,
    SERIAL_OPT_DUMMY = 0x010A
};
```

Constants

`SERIAL_OPT_BAUDRATE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`SERIAL_OPT_DATABITS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`SERIAL_OPT_STOPBITS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`SERIAL_OPT_PARITY`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`SERIAL_OPT_STATUS`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`SERIAL_OPT_HANDSHAKE`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`SERIAL_OPT_RCVTIMEOUT`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`SERIAL_OPT_ERRORCHARACTER`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`SERIAL_OPT_EXTCLOCK`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`SERIAL_OPT_BURSTMODE`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`SERIAL_OPT_DUMMY`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

SIGHUP

```
enum {
    SIGHUP = 1,
    SIGURG = 16,
    SIGPOLL = 30
};
```

Constants
SIGHUP
SIGURG
SIGPOLL

SL_FATAL

```
enum {
    SL_FATAL = 0x01,
    SL_NOTIFY = 0x02,
    SL_ERROR = 0x04,
    SL_TRACE = 0x08,
    SL_CONSOLE = 0x00,
    SL_WARN = 0x20,
    SL_NOTE = 0x40
};
```

Constants

SL_FATAL
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProtocol.h.

SL_NOTIFY
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProtocol.h.

SL_ERROR
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProtocol.h.

SL_TRACE
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProtocol.h.

SL_CONSOLE
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProtocol.h.

SL_WARN
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProtocol.h.

SL_NOTE
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProtocol.h.

SNDZERO

```
enum {
    SNDZERO = 0x01
};
```

Constants

`SNDZERO`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

SO_ALL

```
enum {
    SO_ALL = 0x7FFF,
    SO_READOPT = 0x0001,
    SO_WROFF = 0x0002,
    SO_MINPSZ = 0x0004,
    SO_MAXPSZ = 0x0008,
    SO_HIWAT = 0x0010,
    SO_LOWAT = 0x0020,
    SO_MREADON = 0x0040,
    SO_MREADOFF = 0x0080,
    SO_NDELON = 0x0100,
    SO_NDELOFF = 0x0200,
    SO_ISTTY = 0x0400,
    SO_ISNTTY = 0x0800,
    SO_TOSTOP = 0x1000,
    SO_TONSTOP = 0x2000,
    SO_BAND = 0x4000,
    SO_POLL_SET = 0x8000,
    SO_POLL_CLR = 0x00010000
};
```

Constants

`SO_ALL`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`SO_READOPT`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`SO_WROFF`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`SO_MINPSZ`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`SO_MAXPSZ`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProtocol.h`.

`SO_HIWAT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`SO_LOWAT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`SO_MREADON`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`SO_MREADOFF`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`SO_NDELON`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`SO_NDELOFF`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`SO_ISTTY`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`SO_ISNTTY`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`SO_TOSTOP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`SO_TONSTOP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`SO_BAND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`SO_POLL_SET`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`SO_POLL_CLR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

SQLVL_QUEUE

```
enum {
    SQLVL_QUEUE = 1,
    SQLVL_QUEUEPAIR = 2,
    SQLVL_MODULE = 3,
    SQLVL_GLOBAL = 4,
    SQLVL_DEFAULT = 3
};
```

Constants
SQLVL_QUEUE
SQLVL_QUEUEPAIR
SQLVL_MODULE
SQLVL_GLOBAL
SQLVL_DEFAULT

STRCANON

```
enum {
    STRCANON = 0x01,
    RECOPY = 0x02
};
```

Constants
STRCANON
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProtocol.h.
RECOPY
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProtocol.h.

STRCTLSZ

```
enum {
    STRCTLSZ = 256,
    STRMSGSZ = 8192
};
```

Constants
STRCTLSZ
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProtocol.h.
STRMSGSZ
Available in Mac OS X v10.0 and later.
Declared in OpenTransportProtocol.h.

T_ADDR

```
enum {
    T_ADDR = 0x01,
    T_OPT = 0x02,
    T_UDATA = 0x04,
    T_ALL = 0xFFFF
};
typedef UInt32 OTFieldsType;
```

Constants

`T_ADDR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`T_OPT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`T_UDATA`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`T_ALL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

T_ATALKBADROUTEREVENT

```
enum {
    T_ATALKBADROUTEREVENT = kAppleTalkEvent + 70,
    T_ALLNODESTAKENEVENT = kAppleTalkEvent + 71,
    T_FIXEDNODETAKENEVENT = kAppleTalkEvent + 72,
    T_MPPCOMPATCFIGEVENT = kAppleTalkEvent + 73,
    T_FIXEDNODEBADEVENT = kAppleTalkEvent + 74
};
```

Constants

`T_ATALKBADROUTEREVENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_ALLNODESTAKENEVENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_FIXEDNODETAKENEVENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_MPPCOMPATCFIGEVENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_FIXEDNODEBADEVENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

Structure Types

Specify the various Open Transport structure types that can be allocated by the `OTAllocInContext` function.

```
enum {
    T_BIND = 1,
    T_OPTMGMT = 2,
    T_CALL = 3,
    T_DIS = 4,
    T_UNITDATA = 5,
    T_UDERROR = 6,
    T_INFO = 7,
    T_REPLYDATA = 8,
    T_REQUESTDATA = 9,
    T_UNITREQUEST = 10,
    T_UNITREPLY = 11
};
typedef UInt32 OTStructType;
```

Constants

`T_BIND`
Specifies the `TBind` (page 2540) structure.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`T_OPTMGMT`
Specifies the `The Option Management Structure` (page 2549) structure.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`T_CALL`
Specifies the `TCall` (page 2541) structure
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`T_DIS`
Specifies the `TDiscon` (page 2542) structure.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`T_UNITDATA`
Specifies the `TUnitData` (page 2553) structure.
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`T_UDERROR`
Specifies the `TUDErr` (page 2552) structure.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_INFO`
Speciifies the `TEndpointInfo` (page 2542) structure.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_REPLYDATA`
Specifies the `TReply` (page 2552) structure.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_REQUESTDATA`
Specifies the `TRequest` (page 2552) structure.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_UNITREQUEST`
Specifies the `TUnitRequest` (page 2555) structure.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_UNITREPLY`
Specifies the `TUnitReply` (page 2554) structure.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

T_DNRSTRINGTOADDRCOMPLETE

```
enum {
    T_DNRSTRINGTOADDRCOMPLETE = kPRIVATEEVENT + 1,
    T_DNRADDRTONAMECOMPLETE = kPRIVATEEVENT + 2,
    T_DNRSYSINFOCOMPLETE = kPRIVATEEVENT + 3,
    T_DNRMAILEXCHANGECOMPLETE = kPRIVATEEVENT + 4,
    T_DNRQUERYCOMPLETE = kPRIVATEEVENT + 5
};
```

Constants

`T_DNRSTRINGTOADDRCOMPLETE`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`T_DNRADDRTONAMECOMPLETE`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`T_DNRSYSINFOCOMPLETE`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`T_DNRMAILEXCHANGECOMPLETE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_DNRQUERYCOMPLETE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

T_GARBAGE

```
enum {
    T_GARBAGE = 2
};
```

Constants
`T_GARBAGE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

T_INFINITE

```
enum {
    T_INFINITE = -1,
    T_INVALID = -2
};
```

Constants
`T_INFINITE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`T_INVALID`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

Event Codes

Define the constant names that provider functions can use for event codes, or define port-related events Open Transport can send to an client application.

```
typedef UInt32 OTEventCode;
enum {
    T_LISTEN = 0x0001,
    T_CONNECT = 0x0002,
    T_DATA = 0x0004,
    T_EXDATA = 0x0008,
    T_DISCONNECT = 0x0010,
    T_ERROR = 0x0020,
    T_UDERR = 0x0040,
    T_ORDREL = 0x0080,
    T_GODATA = 0x0100,
    T_GOEXDATA = 0x0200,
    T_REQUEST = 0x0400,
    T_REPLY = 0x0800,
    T_PASSCON = 0x1000,
    T_RESET = 0x2000,
    kPRIVATEEVENT = 0x10000000,
    kCOMPLETEEVENT = 0x20000000,
    T_BINDCOMPLETE = 0x20000001,
    T_UNBINDCOMPLETE = 0x20000002,
    T_ACCEPTCOMPLETE = 0x20000003,
    T_REPLYCOMPLETE = 0x20000004,
    T_DISCONNECTCOMPLETE = 0x20000005,
    T_OPTMGMTCOMPLETE = 0x20000006,
    T_OPENCOMPLETE = 0x20000007,
    T_GETPROTADDRCOMPLETE = 0x20000008,
    T_RESOLVEADDRCOMPLETE = 0x20000009,
    T_GETINFOCOMPLETE = 0x2000000A,
    T_SYNCCOMPLETE = 0x2000000B,
    T_MEMORYRELEASED = 0x2000000C,
    T_REGNAMECOMPLETE = 0x2000000D,
    T_DELNAMECOMPLETE = 0x2000000E,
    T_LKUPNAMECOMPLETE = 0x2000000F,
    T_LKUPNAMERESULT = 0x20000010,
    kOTSyncIdleEvent = 0x20000011,
    kSTREAMEVENT = 0x21000000,
    kOTReservedEvent1 = 0x21000001,
    kGetmsgEvent = 0x21000002,
    kStreamReadEvent = 0x21000003,
    kStreamWriteEvent = 0x21000004,
    kStreamIoctlEvent = 0x21000005,
    kOTReservedEvent2 = 0x21000006,
    kStreamOpenEvent = 0x21000007,
    kPollEvent = 0x21000008,
    kOTReservedEvent3 = 0x21000009,
    kOTReservedEvent4 = 0x2100000A,
    kOTReservedEvent5 = 0x2100000B,
    kOTReservedEvent6 = 0x2100000C,
    kOTReservedEvent7 = 0x2100000D,
    kOTReservedEvent8 = 0x2100000E,
    kSIGNALEVENT = 0x22000000,
    kPROTOCOLEVENT = 0x23000000,
    kOTProviderIsDisconnected = 0x23000001,
    kOTProviderIsReconnected = 0x23000002,
    kOTProviderWillClose = 0x24000001,
    kOTProviderIsClosed = 0x24000002,
    kOTPortDisabled = 0x25000001,
    kOTPortEnabled = 0x25000002,
```

```
    kOTPortOffline = 0x25000003,
    kOTPortOnline = 0x25000004,
    kOTClosePortRequest = 0x25000005,
    kOTYieldPortRequest = 0x25000005,
    kOTNewPortRegistered = 0x25000006,
    kOTPortNetworkChange = 0x25000007,
    kOTConfigurationChanged = 0x26000001,
    kOTSystemSleep = 0x26000002,
    kOTSystemShutdown = 0x26000003,
    kOTSystemAwaken = 0x26000004,
    kOTSystemIdle = 0x26000005,
    kOTSystemSleepPrep = 0x26000006,
    kOTSystemShutdownPrep = 0x26000007,
    kOTSystemAwakenPrep = 0x26000008,
    kOTStackIsLoading = 0x27000001,
    kOTStackWasLoaded = 0x27000002,
    kOTStackIsUnloading = 0x27000003
};

/*   The following event codes are used internally by Open Transport.*/
enum {
    kOTDisablePortEvent = 0x21000001,
    kStreamCloseEvent = 0x21000006,
    kBackgroundStreamEvent = 0x21000009,
    kIoctlRecvFdEvent = 0x2100000A,
    kOTTryShutdownEvent = 0x2100000B,
    kOTScheduleTerminationEvent = 0x2100000C,
    kOTEnablePortEvent = 0x2100000D,
    kOTNewPortRegisteredEvent = 0x2100000E,
    kOTPortOfflineEvent = 0x2100000F,
    kOTPortOnlineEvent = 0x21000010,
    kOTPortNetworkChangeEvent = 0x21000011
};
```

Constants

`T_LISTEN`

A connection request has arrived. Call the `OTListen` function to read the request.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_CONNECT`

The passive peer has accepted a connection that you requested using the `OTConnect` function. Call the `OTRcvConnect` function to retrieve any data or option information that the passive peer has specified when accepting the connection or to retrieve the address to which you are actually connected. The `cookie` parameter to the notifier function is the `sndCall` parameter that you specified when calling the `OTConnect` function.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_DATA`

Normal data has arrived. Depending on the type of service of the endpoint you are using, you can call the `OTRcvUData` function or the `OTRcv` function to read it. Continue reading data until the function returns with the `kOTNoDataErr` result; you will not get another indication that data has arrived until you have read all the data and got this error.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_EXDATA`

Expedited data has arrived. Use the `OTRcv` function to read it. Continue reading data by calling the `OTRcv` function until the function returns with the `kOTNoDataErr` result; you will not get another indication that data has arrived until you have read all the data and got this error.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_DISCONNECT`

A connection has been torn down or rejected. Use the `OTRcvDisconnect` function to clear the event.

If the event is used to signify that a connection has been terminated, the cookie parameter to the notifier is `NULL`.

If the event indicates a rejected connection request, the `cookie` parameter to the notification routine is the same as the `sndCall` parameter that you passed to the `OTConnect` function.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_ERROR`

Obsolete.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_UDERR`

The provider was not able to send the data you specified using the `OTSndUData` function even though the function returned successfully. You must call the `OTRcvUDErr` function to clear this event and determine why the function failed.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_ORDREL`

The remote client has called the `OTSndOrderlyDisconnect` function to initiate an orderly disconnect. You must call the `OTRcvOrderlyDisconnect` function to acknowledge receiving the event.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_GODATA`

Flow-control restrictions have been lifted. You can now send normal data.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_GOEXDATA`

Flow-control restrictions have been lifted. You can now send expedited data.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_REQUEST`

A request has arrived. Depending on the type of service for the endpoint you are using, you can call the `OTRcvRequest` function or the `OTRcvURequest` function to receive it. You must continue to call the function until it returns with the `kOTNoDataErr` result.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_REPLY`

A response to a request has arrived. Depending on the type of service of the endpoint you are using, you can call the `OTRcvReply` function or `OTRcvUReply` function to receive it. You must continue to call the function until it returns with the `kOTNoDataErr` result.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_PASSCON`

When the `OTAccept` function completes, the endpoint provider passes this event to the endpoint receiving the connection (whether that endpoint is the same as or different from the endpoint that calls the `OTAccept` function). The `cookie` parameter contains the `resRef` parameter to the `OTAccept` function.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_RESET`

A connection-oriented endpoint has received a reset from the remote end and has flushed all unread and unsent data. This only occurs for some types of endpoints, and it generally leaves the endpoint in an unknown state.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kPRIVATEEVENT`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kCOMPLETEEVENT`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_BINDCOMPLETE`

The `OTBind` function has completed. The `cookie` parameter contains the `retAddr` parameter of the bind call.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_UNBINDCOMPLETE`

The `OTUnbind` function has completed. The `cookie` parameter is meaningless.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_ACCEPTCOMPLETE`

The `OTAccept` function has completed. The `cookie` parameter contains the `resRef` parameter to the `OTAccept` function.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_REPLYCOMPLETE`
The `OTSndUReply` or `OTSndReply` functions have completed. The `cookie` parameter contains the sequence number used in the `OTSndUReply` or `OTSndReply` call.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_DISCONNECTCOMPLETE`
The `OTSndDisconnect` function has completed. The `cookie` parameter contains the call parameter of the `OTSndDisconnect` function.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_OPTMGMTCOMPLETE`
The `OTOptionManagement` function has completed. The `cookie` parameter contains the `ret` parameter that you passed to the function.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_OPENCOMPLETE`
An asynchronous call to open a provider has completed. The `cookie` parameter contains the provider reference.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_GETPROTADDRCOMPLETE`
The `OTGetProtAddress` function has completed. The `cookie` parameter contains the `peerAddr` parameter that you passed to the `OTGetProtocolAddress` function. If you passed `NULL` for that parameter, the `cookie` parameter contains the address passed in the `boundAddr` parameter.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_RESOLVEADDRCOMPLETE`
The `OTResolveAddress` function has completed. The `cookie` parameter contains the `retAddr` parameter of the `OTResolveAddress` function.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_GETINFOCOMPLETE`
The `OTGetEndpointInfo` function has completed. The `cookie` parameter contains the `info` parameter of the `OTGetEndpointInfo` function.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_SYNCCOMPLETE`
The `OTSync` function has completed. The `cookie` parameter is meaningless.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_MEMORYRELEASED`

You are using an asynchronous endpoint that acknowledges sends and Open Transport is done using the buffers containing the data you are sending. If you called the `OTSnd` function, the `cookie` parameter contains the `buf` parameter. If you called the `OTSndUData` function, the `cookie` parameter contains the `udata` parameter. The `result` parameter contains the number of bytes that were sent. This might be less than the number you meant to send due to flow-control or memory restrictions.

You should not wait for the `T_MEMORYRELEASED` event from a previous send operation to trigger more sends. The exact time this event occurs depends on how the underlying provider is implemented. It might hold on to memory until the next send occurs, or behave in some other way which causes it to delay releasing memory.

Note that `T_MEMORYRELEASED` events can reenter your notifier. See `OTAckSends` (page 2307)for more information.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_REGNAMECOMPLETE`

The `OTRegisterName` function has completed. The `cookie` parameter is the reply parameter, unless it was `NULL`. In this case, it is the `req` parameter.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_DELNAMECOMPLETE`

The `OTDeleteName` function or the `OTDeleteNameByID` function has completed. The `cookie` parameter contains the `name` parameter or the `nameId` parameter of the function, respectively.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_LKUPNAMECOMPLETE`

The `OTLookupName` function has completed. The `cookie` parameter contains the `reply` parameter of the `OTLookUpName` function.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_LKUPNAMERESULT`

An `OTLookupName` function has found a name and is returning it, but the lookup is not yet complete. The `cookie` parameter contains the `reply` parameter passed to the `OTLookupName` function.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTSyncIdleEvent`

A synchronous call is waiting to complete.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kSTREAMEVENT`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTReservedEvent1`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kGetmsgEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kStreamReadEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kStreamWriteEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kStreamIoctlEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTReservedEvent2`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kStreamOpenEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kPollEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTReservedEvent3`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTReservedEvent4`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTReservedEvent5`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTReservedEvent6`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTReservedEvent7`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTReservedEvent8`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kSIGNALEVENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kPROTOCOLEVENT`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTProviderIsDisconnected`

Your provider was bound with a `qlen` parameter value greater than 0 and it has been disconnected (is no longer listening). You receive this event after a port has accepted a request to temporarily yield ownership of a port to another provider, which causes this provider to be disconnected from the port in question. This currently only happens with serial ports, but could also happen with other connection-oriented drivers that have characteristics similar to serial ports. You get a `kOTProviderIsReconnected` message when the port reverts back to this provider's ownership.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTProviderIsReconnected`

Your provider has been reconnected, that is, the cause for its disconnection has been relieved.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTProviderWillClose`

When you return from the notifier function, Open Transport will close the provider whose reference is contained in the `cookie` parameter. The `result` parameter contains a result code specifying the reason why the provider had to close. For example, the user may have decided to switch links using the TCP/IP or AppleTalk control panel. The result codes that can be returned are in the range –3280 through –3285; these are documented in "Open Transport Result Codes" (page 2722).

You can only get this event at system task time. Consequently, you are allowed to set the endpoint to synchronous mode (from within the notifier function) and call functions synchronously before you return from the notifier, at which point the provider is closed. After this, any calls other than `OTCloseProvider` will fail with a `kOTOutStateErr` error.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTProviderIsClosed`

The provider has closed. The reason for being closed can be found in the `OTResult` value passed to your notifier. The reasons typically are `kOTPortHasDiedErr`, `kOTPortWasEjectedErr`, or `kOTPortLostConnectionErr`. At this point, any calls other than `OTCloseProvider` will fail with a `kOTOutStateErr` error.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTPortDisabled`

A port has gone off line, as when the user removes a PCMCIA card while the computer is running. The OTResult parameter specifies the reason, if known, and the cookie parameter provides the port reference of the port that went off line. A port going off line often results in providers getting kOTProviderIsClosed events. There is no guarantee in Open Transport as to which of these events will be received first.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTPortEnabled`
A port that had previously been disabled is now reenabled, as when the user reinserts a previously removed PCMCIA card while the computer is running. The cookie parameter is the port reference of the port that is now enabled.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTPortOffline`
The port is now offline.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTPortOnline`
A request has been made to close or yield this port.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTClosePortRequest`
You currently are using a provider that is using a port that some other application wants to use. The OTResult parameter is the reason for the request (normally kOTNoError or kOTUserRequestedErr), and the cookie parameter is a pointer to an OTPortCloseStruct structure.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTYieldPortRequest`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTNewPortRegistered`
A new port has been registered with Open Transport, as when the user inserts a new PCMCIA card. The cookie parameter is the port reference of the new port. Your provider receives this event the first time a new port is enabled. Subsequently, if a port is reenabled after being disabled, you receive the kOTPortEnabled event instead.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTPortNetworkChange`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTConfigurationChanged`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTSystemSleep`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTSystemShutdown`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kOTSystemAwaken`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTSystemIdle`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTSystemSleepPrep`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTSystemShutdownPrep`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTSystemAwakenPrep`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTStackIsLoading`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTStackWasLoaded`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTStackIsUnloading`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTDisablePortEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kStreamCloseEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kBackgroundStreamEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kIoctlRecvFdEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTTryShutdownEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTScheduleTerminationEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTEnablePortEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTNewPortRegisteredEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPortOfflineEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPortOnlineEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`kOTPortNetworkChangeEvent`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

Open Transport Flags and Status Codes

Specify information about data transmitted with the `OTSnd` or `OTRcv` functions, or specify options for the `OTOptionManagement` function, or indicate the result status of an option negotiation.

```
typedef UInt32 OTFlags;
/*   These flags are used when sending and receiving data.The constants
defined are masks.*/
enum {
    T_MORE = 0x0001,
    T_EXPEDITED = 0x0002,
    T_ACKNOWLEDGED = 0x0004,
    T_PARTIALDATA = 0x0008,
    T_NORECEIPT = 0x0010,
    T_TIMEDOUT = 0x0020
};

/* These flags are used in the TOptMgmt structure to request services.*/
enum {
    T_NEGOTIATE = 0x0004,
    T_CHECK = 0x0008,
    T_DEFAULT = 0x0010,
    T_CURRENT = 0x0080
};
```

```
/*   These flags are used in the TOptMgmt and TOption structures
to   return results.*/
enum {
    T_SUCCESS = 0x0020,
    T_FAILURE = 0x0040,
    T_PARTSUCCESS = 0x0100,
    T_READONLY = 0x0200,
    T_NOTSUPPORT = 0x0400
};
```

Constants

`T_MORE`

There is more data for the current TSDU or ETSDU. The next send or receive operation will handle additional data for this TSDU or ETSDU.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_EXPEDITED`

On sends, the data is sent as expedited data if the endpoint supports expedited data. On receives, the flag indicates that expedited data was sent.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_ACKNOWLEDGED`

The transaction must be acknowledged before the send or receive function can complete.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_PARTIALDATA`

There is more data for the current TSDU or ETSDU. Unlike `T_MORE`, `T_PARTIALDATA` does not guarantee that the next send or receive operation will handle additional data for this TSDU or ETSDU.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_NORECEIPT`

There is no need to send a `T_REPLY_COMPLETE` event to complete the transaction. If you don't need to know when the transaction is actually done, you can set this flag to improve performance.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_TIMEDOUT`

The reply timed out. If a protocol such as ATP loses the acknowledgment for a transaction that needs to be acknowledged, the transaction will eventually time out. Since the reply didn't really fail (it just timed out), Open Transport can send a `T_REPLY_COMPLETE` event to complete the transaction and set this flag to explain what happened.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_NEGOTIATE`
: Negotiate the option values specified in the opt.buf field of the req parameter.

  The overall result of the negotiation is specified by the flags field of the ret parameter. The opt.buf field of the ret parameter points to a buffer where negotiated values for each option are placed.

  Available in Mac OS X v10.0 and later.

  Declared in `OpenTransport.h`.

`T_CHECK`
: Verify whether the endpoint supports the options referenced by the opt.buf field of the req parameter. The overall result of the verification is specified by the flags field of the ret parameter. Specific verification results are returned in the opt.buf field of the ret parameter.

  Available in Mac OS X v10.0 and later.

  Declared in `OpenTransport.h`.

`T_DEFAULT`
: Retrieve the default value for those options in the buffer referenced by the req->opt.buf field. To retrieve default values for all the options supported by an endpoint, include just the option T_ALLOPT in the options buffer.Option values are returned in the opt.buf field of the ret parameter.

  Available in Mac OS X v10.0 and later.

  Declared in `OpenTransport.h`.

`T_CURRENT`
: Retrieve the current value for those options that the endpoint supports and that are specified in the buffer referenced by the req->opt.buf field. To retrieve current values for all the options that an endpoint supports, include just the option T_ALLOPT in the options buffer.Option values are returned in the opt.buf field of the ret parameter.

  Available in Mac OS X v10.0 and later.

  Declared in `OpenTransport.h`.

`T_SUCCESS`
: The requested value was negotiated.

  Available in Mac OS X v10.0 and later.

  Declared in `OpenTransport.h`.

`T_FAILURE`
: The negotiation failed.

  Available in Mac OS X v10.0 and later.

  Declared in `OpenTransport.h`.

`T_PARTSUCCESS`
: A lower requested value was negotiated.

  Available in Mac OS X v10.0 and later.

  Declared in `OpenTransport.h`.

`T_READONLY`
: The option was read-only.

  Available in Mac OS X v10.0 and later.

  Declared in `OpenTransport.h`.

`T_NOTSUPPORT`
The endpoint does not support the requested value.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

T_NOTOS

```
enum {
    T_NOTOS = 0x00,
    T_LDELAY = (1 << 4),
    T_HITHRPT = (1 << 3),
    T_HIREL = (1 << 2)
};
```

Constants

`T_NOTOS`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`T_LDELAY`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`T_HITHRPT`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`T_HIREL`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

T_NULL

```
enum {
    T_NULL = NULL,
    T_UNSPEC = -3
};
```

Constants

`T_NULL`

`T_UNSPEC`
The option does not have a fully specified value at this time. An endpoint provider might return this status code if it cannot currently access the option value. This might happen if the endpoint is in the state T_UNBND in systems where the protocol stack resides on a separate host.

T_ROUTINE

```
enum {
    T_ROUTINE = 0,
    T_PRIORITY = 1,
    T_IMMEDIATE = 2,
    T_FLASH = 3,
    T_OVERRIDEFLASH = 4,
    T_CRITIC_ECP = 5,
    T_INETCONTROL = 6,
    T_NETCONTROL = 7
};
```

Constants

`T_ROUTINE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_PRIORITY`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_IMMEDIATE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_FLASH`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_OVERRIDEFLASH`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_CRITIC_ECP`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_INETCONTROL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`T_NETCONTROL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

Endpoint Flags

Specifies information about an endpoint.

```
enum {
    T_SENDZERO = 0x0001,
    T_XPG4_1 = 0x0002,
    T_CAN_SUPPORT_MDATA = 0x10000000,
    T_CAN_RESOLVE_ADDR = 0x40000000,
    T_CAN_SUPPLY_MIB = 0x20000000
};
```

Constants

`T_SENDZERO`

This endpoint lets you send and receive zero-length TSDUs.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_XPG4_1`

This endpoint supports the `OTGetProtAddress` function (conforms to XTI in XPG4).

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_CAN_SUPPORT_MDATA`

This endpoint supports `M_DATA`, that is, it permits receiving and returning raw packets.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_CAN_RESOLVE_ADDR`

This endpoint supports the `OTResolveAddress` function.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_CAN_SUPPLY_MIB`

This endpoint can supply the Management Information Base (MIB) data used by the Simple Network Management Protocol (SNMP). At this time you cannot access this data.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

T_UNSPEC

Constants

T_YES

```
enum {
    T_YES = 1,
    T_NO = 0,
    T_UNUSED = -1,
    kT_NULL = 0,
    T_ABSREQ = 0x8000
};
```

Constants

`T_YES`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_NO`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_UNUSED`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`kT_NULL`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`T_ABSREQ`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

TCP_NODELAY

```
enum {
    TCP_NODELAY = 0x01,
    TCP_MAXSEG = 0x02,
    TCP_NOTIFY_THRESHOLD = 0x10,
    TCP_ABORT_THRESHOLD = 0x11,
    TCP_CONN_NOTIFY_THRESHOLD = 0x12,
    TCP_CONN_ABORT_THRESHOLD = 0x13,
    TCP_OOBINLINE = 0x14,
    TCP_URGENT_PTR_TYPE = 0x15,
    TCP_KEEPALIVE = 0x0008
};
```

Constants

`TCP_NODELAY`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`TCP_MAXSEG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`TCP_NOTIFY_THRESHOLD`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`TCP_ABORT_THRESHOLD`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`TCP_CONN_NOTIFY_THRESHOLD`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`TCP_CONN_ABORT_THRESHOLD`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`TCP_OOBINLINE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`TCP_URGENT_PTR_TYPE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

`TCP_KEEPALIVE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProviders.h`.

TE_OPENED

```
enum {
    TE_OPENED = 1,
    TE_BIND = 2,
    TE_OPTMGMT = 3,
    TE_UNBIND = 4,
    TE_CLOSED = 5,
    TE_CONNECT1 = 6,
    TE_CONNECT2 = 7,
    TE_ACCEPT1 = 8,
    TE_ACCEPT2 = 9,
    TE_ACCEPT3 = 10,
    TE_SND = 11,
    TE_SNDDIS1 = 12,
    TE_SNDDIS2 = 13,
    TE_SNDREL = 14,
    TE_SNDUDATA = 15,
    TE_LISTEN = 16,
    TE_RCVCONNECT = 17,
    TE_RCV = 18,
    TE_RCVDIS1 = 19,
    TE_RCVDIS2 = 20,
    TE_RCVDIS3 = 21,
    TE_RCVREL = 22,
    TE_RCVUDATA = 23,
    TE_RCVUDERR = 24,
    TE_PASS_CONN = 25,
    TE_BAD_EVENT = 26
};
```

Constants

`TE_OPENED`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_BIND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_OPTMGMT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_UNBIND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_CLOSED`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_CONNECT1`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_CONNECT2`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_ACCEPT1`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_ACCEPT2`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_ACCEPT3`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_SND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_SNDDIS1`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_SNDDIS2`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_SNDREL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_SNDUDATA`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_LISTEN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_RCVCONNECT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_RCV`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_RCVDIS1`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_RCVDIS2`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_RCVDIS3`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_RCVREL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_RCVUDATA`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_RCVUDERR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_PASS_CONN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TE_BAD_EVENT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

TS_UNBND

```
enum {
    TS_UNBND = 1,
    TS_WACK_BREQ = 2,
    TS_WACK_UREQ = 3,
    TS_IDLE = 4,
    TS_WACK_OPTREQ = 5,
    TS_WACK_CREQ = 6,
    TS_WCON_CREQ = 7,
    TS_WRES_CIND = 8,
    TS_WACK_CRES = 9,
    TS_DATA_XFER = 10,
    TS_WIND_ORDREL = 11,
    TS_WREQ_ORDREL = 12,
    TS_WACK_DREQ6 = 13,
    TS_WACK_DREQ7 = 14,
    TS_WACK_DREQ9 = 15,
    TS_WACK_DREQ10 = 16,
    TS_WACK_DREQ11 = 17,
    TS_WACK_ORDREL = 18,
    TS_NOSTATES = 19,
    TS_BAD_STATE = 19
};
```

Constants

`TS_UNBND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_WACK_BREQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_WACK_UREQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_IDLE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_WACK_OPTREQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_WACK_CREQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_WCON_CREQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_WRES_CIND`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_WACK_CRES`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_DATA_XFER`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_WIND_ORDREL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_WREQ_ORDREL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_WACK_DREQ6`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_WACK_DREQ7`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_WACK_DREQ9`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_WACK_DREQ10`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_WACK_DREQ11`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_WACK_ORDREL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_NOSTATES`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

`TS_BAD_STATE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransportProtocol.h`.

TSUCCESS

```
typedef UInt16 OTXTIErr;
enum {
    TSUCCESS = 0,
    TBADADDR = 1,
    TBADOPT = 2,
    TACCES = 3,
    TBADF = 4,
    TNOADDR = 5,
    TOUTSTATE = 6,
    TBADSEQ = 7,
    TSYSERR = 8,
    TLOOK = 9,
    TBADDATA = 10,
    TBUFOVFLW = 11,
    TFLOW = 12,
    TNODATA = 13,
    TNODIS = 14,
    TNOUDERR = 15,
    TBADFLAG = 16,
    TNOREL = 17,
    TNOTSUPPORT = 18,
    TSTATECHNG = 19,
    TNOSTRUCTYPE = 20,
    TBADNAME = 21,
    TBADQLEN = 22,
    TADDRBUSY = 23,
    TINDOUT = 24,
    TPROVMISMATCH = 25,
    TRESQLEN = 26,
    TRESADDR = 27,
    TQFULL = 28,
    TPROTO = 29,
    TBADSYNC = 30,
    TCANCELED = 31,
    TLASTXTIERROR = 31
};
```

Constants

`TSUCCESS`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`TBADADDR`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`TBADOPT`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`TACCES`

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`TBADF`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TNOADDR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TOUTSTATE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TBADSEQ`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TSYSERR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TLOOK`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TBADDATA`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TBUFOVFLW`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TFLOW`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TNODATA`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TNODIS`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TNOUDERR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TBADFLAG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TNOREL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TNOTSUPPORT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TSTATECHNG`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TNOSTRUCTYPE`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TBADNAME`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TBADQLEN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TADDRBUSY`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TINDOUT`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TPROVMISMATCH`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TRESQLEN`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TRESADDR`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TQFULL`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TPROTO`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TBADSYNC`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TCANCELED`
Available in Mac OS X v10.0 and later.
Declared in `OpenTransport.h`.

`TLASTXTIERROR`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

UDP_CHECKSUM

```
enum {
    UDP_CHECKSUM = 0x0600,
    UDP_RX_ICMP = 0x02
};
```

Constants

`UDP_CHECKSUM`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

`UDP_RX_ICMP`
Available in Mac OS X v10.0 and later.

Declared in `OpenTransportProviders.h`.

XTI-Level Options and Generic Options

Specifies constant names for XTI-level options.

```
enum {
    XTI_DEBUG = 0x0001,
    XTI_LINGER = 0x0080,
    XTI_RCVBUF = 0x1002,
    XTI_RCVLOWAT = 0x1004,
    XTI_SNDBUF = 0x1001,
    XTI_SNDLOWAT = 0x1003,
    XTI_PROTOTYPE = 0x1005,
    OPT_CHECKSUM = 0x0600,
    OPT_RETRYCNT = 0x0601,
    OPT_INTERVAL = 0x0602,
    OPT_ENABLEEOM = 0x0603,
    OPT_SELFSEND = 0x0604,
    OPT_SERVERSTATUS = 0x0605,
    OPT_ALERTENABLE = 0x0606,
    OPT_KEEPALIVE = 0x0008
};
```

Constants

`XTI_DEBUG`
A 32 bit constant specifying whether debugging is enabled. Debugging is disabled if the option is specified with no value. This option is an absolute requirement.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`XTI_LINGER`

A value defined by a linger structure (page 571) that specifies whether the option is turned on (T_YES) or off (T_NO) and specifies a linger period in seconds. This option is an absolute requirement; however, you do not have to specify a value for the l_linger field of the linger structure.

You use this option to extend the execution of the OTCloseProvider function for some specified amount of time. The delay allows data still queued in the endpoint's internal send buffer to be sent before the endpoint provider is closed. If you call the OTCloseProvider function and the send buffer is not empty, the endpoint provider attempts to send the remaining data during the linger period, before closing. Open Transport discards any data remaining in the send buffer after the linger period has elapsed.

Consult the documentation for your protocol to determine the valid range of values for the linger period.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`XTI_RCVBUF`

A 32-bit integer specifying the size of the endpoint's internal buffer allocated for receiving data. You can increase the size of this buffer for high-volume connections or decrease the buffer to limit the possible backlog of incoming data.

This option is not an absolute requirement. Consult the documentation for your protocol to determine the valid range of values for the buffer size.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`XTI_RCVLOWAT`

A 32-bit integer specifying the low-water mark for the receive buffer—that is, the number of bytes that must accumulate in the endpoint's internal receive buffer before you are advised that data has arrived via a T_DATA event. Choosing a value that is too low might result in your application's getting an excessive number of T_DATA events and doing unnecessary reads. Choosing a value that is too high might result in Open Transport running out of memory and disabling incoming data packets.

This option is not an absolute requirement. Consult the documentation for your protocol to determine the valid range of values for the low-water mark.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`XTI_SNDBUF`

A 32-bit integer specifying the size of the endpoint's internal buffer allocated for sending data. Specifying a value that is too low might result in Open Transport doing more sends than necessary and wasting processor time; specifying a value that is too high might cause flow control problems.

This option is not an absolute requirement. Consult the documentation for your protocol to determine the valid range of values for the buffer size.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`XTI_SNDLOWAT`

A 32-bit integer specifying the low-water mark for the send buffer—that is, the number of bytes that must accumulate in the endpoint's internal send buffer before Open Transport actually sends the data. Choosing a value that is too low might result in Open Transport's doing too many sends and wasting processor time. Choosing a value that is too high might result in flow control problems. A value that is slightly lower than the largest packet size defined for the endpoint is a good choice.

This option is not an absolute requirement. Consult the documentation for your protocol to determine the valid range of values for the low-water mark.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`XTI_PROTOTYPE`

The protocol type used by the endpoint. The option is supported by the RawIP endpoint.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`OPT_CHECKSUM`

A 32-bit constant specifying whether checksums are performed. Specify 1 to turn the option on and 0 to turn it off. If you turn it on, a checksum is calculated when a packet is sent and recalculated when the packet is received. If the checksum values match, the client receiving the packet can be fairly certain that data has not been corrupted or lost during transmission. If the checksum values don't match, the receiver discards the packet.

This option is usually implemented by the lowest-level protocol, although you might be allowed to set it at a higher level. For example, if you use an ATP endpoint, you can set checksumming at the ATP level, even though it is implemented by the underlying DDP protocol.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`OPT_RETRYCNT`

A 32-bit integer specifying the number of times a function can attempt packet delivery before returning with an error. A value of 0 means that the function should attempt packet delivery an infinite number of times.

This option is usually implemented by connection-oriented endpoints or connectionless transaction-based endpoints to enable reliable delivery of data. Such protocols normally set a default value for this option.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`OPT_INTERVAL`

A 32-bit integer specifying the interval of time in milliseconds that should elapse between attempts to deliver a packet. The number of attempts is defined by the OPT_RETRYCNT option.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`OPT_ENABLEEOM`

An 32-bit integer specifying end-of-message capability. If you set this option, you enable the use of the T_MORE flag with the OTSnd function to mark the end of a logical unit. This option has meaning only for connection-oriented protocols. A value of 0 clears the option; a value of 1 sets it.

This option is not association-related.

Available in Mac OS X v10.0 and later.

Declared in `OpenTransport.h`.

`OPT_SELFSEND`
: A 32-bit integer allowing you to send broadcast packets to yourself. A value of 0 clears the option; a value of 1 sets it.

  Available in Mac OS X v10.0 and later.

  Declared in `OpenTransport.h`.

`OPT_SERVERSTATUS`
: A string that sets the server's status. The maximum length is protocol dependent. For more information, consult the documentation for the protocol you are using.

  Available in Mac OS X v10.0 and later.

  Declared in `OpenTransport.h`.

`OPT_ALERTENABLE`
: A keepalive structure that specifies whether "keep alive" is turned on (T_YES) or off (T_NO) and specifies the timeout period in minutes.

  Connection-oriented protocols can use this option to check that the connection is maintained. If a connection is established but there is no data being transferred, you can specify a time limit within which Open Transport checks to see that the remote end of the connection is still alive. If it is not, Open Transport tears down the connection.

  This option is association-related.

  Available in Mac OS X v10.0 and later.

  Declared in `OpenTransport.h`.

`OPT_KEEPALIVE`
: Enables or disables protocol alerts.

  Available in Mac OS X v10.0 and later.

  Declared in `OpenTransport.h`.

Discussion
Open Transport defines XTI-level options. These options are not association-related. If the protocol you are using supports these options, you can negotiate them while the endpoint is in any state. The protocol level for all of these options is XTI_GENERIC.

Open Transport also defines some generic options that you can use with any protocol that supports them, listed in this enumeration starting with `OPT_CHECKSUM`. The protocol level for each of these options is the same as the name of the protocol that supports them.

XTI_GENERIC

```
enum {
    XTI_GENERIC = 0xFFFF
};
```

Constants
`XTI_GENERIC`
: Available in Mac OS X v10.0 and later.

  Declared in `OpenTransport.h`.

Result Codes

The most common result codes returned by Open Transport are listed below.

Result Code	Value	Description
kOTNoError	0	No Error occurred Available in Mac OS X v10.0 and later.
kOTBadAddressErr	-3150	XTI2OSStatus(TBADADDR) A Bad address was specified Available in Mac OS X v10.0 and later.
kOTBadOptionErr	-3151	XTI2OSStatus(TBADOPT) A Bad option was specified Available in Mac OS X v10.0 and later.
kOTAccessErr	-3152	XTI2OSStatus(TACCES) Missing access permission Available in Mac OS X v10.0 and later.
kOTBadReferenceErr	-3153	XTI2OSStatus(TBADF) Bad provider reference Available in Mac OS X v10.0 and later.
kOTNoAddressErr	-3154	XTI2OSStatus(TNOADDR) No address was specified Available in Mac OS X v10.0 and later.
kOTOutStateErr	-3155	XTI2OSStatus(TOUTSTATE) Call issued in wrong state Available in Mac OS X v10.0 and later.
kOTBadSequenceErr	-3156	XTI2OSStatus(TBADSEQ) Sequence specified does not exist Available in Mac OS X v10.0 and later.
kOTSysErrorErr	-3157	XTI2OSStatus(TSYSERR) A system error occurred Available in Mac OS X v10.0 and later.
kOTLookErr	-3158	XTI2OSStatus(TLOOK) An event occurred - call Look() Available in Mac OS X v10.0 and later.
kOTBadDataErr	-3159	XTI2OSStatus(TBADDATA) An illegal amount of data was specified Available in Mac OS X v10.0 and later.
kOTBufferOverflowErr	-3160	XTI2OSStatus(TBUFOVFLW) Passed buffer not big enough Available in Mac OS X v10.0 and later.
kOTFlowErr	-3161	XTI2OSStatus(TFLOW) Provider is flow-controlled Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kOTNoDataErr	-3162	XTI2OSStatus(TNODATA) No data available for reading Available in Mac OS X v10.0 and later.
kOTNoDisconnectErr	-3163	XTI2OSStatus(TNODIS) No disconnect indication available Available in Mac OS X v10.0 and later.
kOTNoUDErrErr	-3164	XTI2OSStatus(TNOUDERR) No Unit Data Error indication available Available in Mac OS X v10.0 and later.
kOTBadFlagErr	-3165	XTI2OSStatus(TBADFLAG) A Bad flag value was supplied Available in Mac OS X v10.0 and later.
kOTNoReleaseErr	-3166	XTI2OSStatus(TNOREL) No orderly release indication available Available in Mac OS X v10.0 and later.
kOTNotSupportedErr	-3167	XTI2OSStatus(TNOTSUPPORT) Command is not supported Available in Mac OS X v10.0 and later.
kOTStateChangeErr	-3168	XTI2OSStatus(TSTATECHNG) State is changing - try again later Available in Mac OS X v10.0 and later.
kOTNoStructureTypeErr	-3169	XTI2OSStatus(TNOSTRUCTYPE) Bad structure type requested for OTAlloc Available in Mac OS X v10.0 and later.
kOTBadNameErr	-3170	XTI2OSStatus(TBADNAME) A bad endpoint name was supplied Available in Mac OS X v10.0 and later.
kOTBadQLenErr	-3171	XTI2OSStatus(TBADQLEN) A Bind to an in-use addr with qlen > 0 Available in Mac OS X v10.0 and later.
kOTAddressBusyErr	-3172	XTI2OSStatus(TADDRBUSY) Address requested is already in use Available in Mac OS X v10.0 and later.
kOTIndOutErr	-3173	XTI2OSStatus(TINDOUT) Accept failed because of pending listen Available in Mac OS X v10.0 and later.
kOTProviderMismatchErr	-3174	XTI2OSStatus(TPROVMISMATCH) Tried to accept on incompatible endpoint Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kOTResQLenErr	-3175	XTI2OSStatus(TRESQLEN) Available in Mac OS X v10.0 and later.
kOTResAddressErr	-3176	XTI2OSStatus(TRESADDR) Available in Mac OS X v10.0 and later.
kOTQFullErr	-3177	XTI2OSStatus(TQFULL) Available in Mac OS X v10.0 and later.
kOTProtocolErr	-3178	XTI2OSStatus(TPROTO) An unspecified provider error occurred Available in Mac OS X v10.0 and later.
kOTBadSyncErr	-3179	XTI2OSStatus(TBADSYNC) A synchronous call at interrupt time Available in Mac OS X v10.0 and later.
kOTCanceledErr	-3180	XTI2OSStatus(TCANCELED) The command was cancelled Available in Mac OS X v10.0 and later.
kEPERMErr	-3200	Permission denied Available in Mac OS X v10.0 and later.
kENOENTErr	-3201	No such file or directory Available in Mac OS X v10.0 and later.
kOTNotFoundErr	-3201	OT generic not found error Available in Mac OS X v10.0 and later.
kENORSRCErr	-3202	No such resource Available in Mac OS X v10.0 and later.
kEINTRErr	-3203	Interrupted system service Available in Mac OS X v10.0 and later.
kEIOErr	-3204	I/O error Available in Mac OS X v10.0 and later.
kENXIOErr	-3205	No such device or address Available in Mac OS X v10.0 and later.
kEBADFErr	-3208	Bad file number Available in Mac OS X v10.0 and later.
kEAGAINErr	-3210	Try operation again later Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kENOMEMErr	-3211	Not enough space Available in Mac OS X v10.0 and later.
kOTOutOfMemoryErr	-3211	OT ran out of memory, may be a temporary Available in Mac OS X v10.0 and later.
kEACCESErr	-3212	Permission denied Available in Mac OS X v10.0 and later.
kEFAULTErr	-3213	Bad address Available in Mac OS X v10.0 and later.
kEBUSYErr	-3215	Device or resource busy Available in Mac OS X v10.0 and later.
kEEXISTErr	-3216	File exists Available in Mac OS X v10.0 and later.
kOTDuplicateFoundErr	-3216	OT generic duplicate found error Available in Mac OS X v10.0 and later.
kENODEVErr	-3218	No such device Available in Mac OS X v10.0 and later.
kEINVALErr	-3221	Invalid argument Available in Mac OS X v10.0 and later.
kENOTTYErr	-3224	Not a character device Available in Mac OS X v10.0 and later.
kEPIPEErr	-3231	Broken pipe Available in Mac OS X v10.0 and later.
kERANGEErr	-3233	Message size too large for STREAM Available in Mac OS X v10.0 and later.
kEDEADLKErr	-3234	or a deadlock would occur Available in Mac OS X v10.0 and later.
kEWOULDBLOCKErr	-3234	Call would block, so was aborted Available in Mac OS X v10.0 and later.
kEALREADYErr	-3236	Available in Mac OS X v10.0 and later.
kENOTSOCKErr	-3237	Socket operation on non-socket Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kEDESTADDRREQErr	-3238	Destination address required Available in Mac OS X v10.0 and later.
kEMSGSIZEErr	-3239	Message too long Available in Mac OS X v10.0 and later.
kEPROTOTYPEErr	-3240	Protocol wrong type for socket Available in Mac OS X v10.0 and later.
kENOPROTOOPTErr	-3241	Protocol not available Available in Mac OS X v10.0 and later.
kEPROTONOSUPPORTErr	-3242	Protocol not supported Available in Mac OS X v10.0 and later.
kESOCKTNOSUPPORTErr	-3243	Socket type not supported Available in Mac OS X v10.0 and later.
kEOPNOTSUPPErr	-3244	Operation not supported on socket Available in Mac OS X v10.0 and later.
kEADDRINUSEErr	-3247	Address already in use Available in Mac OS X v10.0 and later.
kEADDRNOTAVAILErr	-3248	Can't assign requested address Available in Mac OS X v10.0 and later.
kENETDOWNErr	-3249	Network is down Available in Mac OS X v10.0 and later.
kENETUNREACHErr	-3250	Network is unreachable Available in Mac OS X v10.0 and later.
kENETRESETErr	-3251	Network dropped connection on reset Available in Mac OS X v10.0 and later.
kECONNABORTEDErr	-3252	Software caused connection abort Available in Mac OS X v10.0 and later.
kECONNRESETErr	-3253	Connection reset by peer Available in Mac OS X v10.0 and later.
kENOBUFSErr	-3254	No buffer space available Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kEISCONNErr	-3255	Socket is already connected Available in Mac OS X v10.0 and later.
kENOTCONNErr	-3256	Socket is not connected Available in Mac OS X v10.0 and later.
kESHUTDOWNErr	-3257	Can't send after socket shutdown Available in Mac OS X v10.0 and later.
kETOOMANYREFSErr	-3258	Too many references: can't splice Available in Mac OS X v10.0 and later.
kETIMEDOUTErr	-3259	Connection timed out Available in Mac OS X v10.0 and later.
kECONNREFUSEDErr	-3260	Connection refused Available in Mac OS X v10.0 and later.
kEHOSTDOWNErr	-3263	Host is down Available in Mac OS X v10.0 and later.
kEHOSTUNREACHErr	-3264	No route to host Available in Mac OS X v10.0 and later.
kEPROTOErr	-3269	Available in Mac OS X v10.0 and later.
kETIMEErr	-3270	Available in Mac OS X v10.0 and later.
kENOSRErr	-3271	Available in Mac OS X v10.0 and later.
kEBADMSGErr	-3272	Available in Mac OS X v10.0 and later.
kECANCELErr	-3273	Available in Mac OS X v10.0 and later.
kENOSTRErr	-3274	Available in Mac OS X v10.0 and later.
kENODATAErr	-3275	Available in Mac OS X v10.0 and later.
kEINPROGRESSErr	-3276	Available in Mac OS X v10.0 and later.
kESRCHErr	-3277	Available in Mac OS X v10.0 and later.
kENOMSGErr	-3278	Available in Mac OS X v10.0 and later.
kOTClientNotInittedErr	-3279	Available in Mac OS X v10.0 and later.
kOTPortHasDiedErr	-3280	Available in Mac OS X v10.0 and later.
kOTPortWasEjectedErr	-3281	Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kOTBadConfigurationErr	-3282	Available in Mac OS X v10.0 and later.
kOTConfigurationChangedErr	-3283	Available in Mac OS X v10.0 and later.
kOTUserRequestedErr	-3284	Available in Mac OS X v10.0 and later.
kOTPortLostConnection	-3285	Available in Mac OS X v10.0 and later.
kModemOutOfMemory	-14000	Available in Mac OS X v10.0 and later.
kModemPreferencesMissing	-14001	Available in Mac OS X v10.0 and later.
kModemScriptMissing	-14002	Available in Mac OS X v10.0 and later.

Search Kit Reference

Framework: CoreServices/CoreServices.h

Declared in SearchKit.h
SKAnalysis.h
SKDocument.h
SKIndex.h
SKSearch.h
SKSummary.h

Overview

Search Kit is a powerful and streamlined C language framework for indexing and searching text in most human languages. It provides fast information retrieval in System Preferences, Address Book, Help Viewer, and Xcode. Apple's Spotlight technology is built on top of Search Kit to provide content searching in Finder, Mail, and the Spotlight menu.

You can use Search Kit or Spotlight to provide similar functionality and powerful information-access capabilities within your Mac OS X application. The Search Kit API is appropriate when you want your application to have full control over indexing and searching, and when your focus is file content. Search Kit is thread-safe and works with Cocoa, Carbon, and command-line tools.

Beginning with Mac OS X version 10.4, Search Kit supports phrase searches, prefix/suffix/substring searches, improved Boolean searches, and improved relevance ranking. Search Kit now uses Spotlight's metadata importers when indexing documents, and takes advantage of any additional importers available on a system. Searching and indexing are much faster with Search Kit's new asynchronous search APIs. And, starting in Mac OS X v10.4, Search Kit provides a summarization API that supplants Find By Content.

Functions by Task

Functions are grouped according to the tasks you perform using them. For an alphabetical list of functions, go to the API index at the end of the document.

Creating, Opening, and Closing Indexes

Search Kit performs its searches not on documents but on its indexes of documents. The functions in this group let your application create memory-based and persistent indexes. Indexes are initially empty. Functions in "Managing Indexes" (page 2730) let you add document content to these indexes.

SKIndexCreateWithURL (page 2747)
Creates a named index in a file whose location is specified with a CFURL object.

SKIndexCreateWithMutableData (page 2746)
Creates a named index stored in a CFMutableData object.

SKIndexOpenWithData (page 2758)
Opens an existing, named index for searching only.

SKIndexOpenWithMutableData (page 2759)
Opens an existing, named index for searching and updating.

SKIndexOpenWithURL (page 2760)
Opens an existing, named index stored in a file whose location is specified with a CFURL object.

SKIndexClose (page 2740)
Closes an index.

SKIndexGetIndexType (page 2754)
Gets the category of an index.

SKIndexGetTypeID (page 2757)
Gets the type identifier for Search Kit indexes.

Managing Indexes

The functions in this section let your application add document content to (and remove document content from) indexes, work with memory- and disk-based indexes, and retrieve metadata from indexes.

SKIndexAddDocumentWithText (page 2739)
Adds a document URL object, and the associated document's textual content, to an index.

SKIndexAddDocument (page 2738)
Adds location information for a file-based document, and the document's textual content, to an index.

SKIndexFlush (page 2750)
Invokes all pending updates associated with an index and commits them to backing store.

SKIndexCompact (page 2741)
Invokes all pending updates associated with an index, compacts the index if compaction is needed, and commits all changes to backing store.

SKIndexGetDocumentCount (page 2751)
Gets the total number of documents represented in an index.

SKIndexGetMaximumDocumentID (page 2755)
Gets the highest-numbered document ID in an index.

SKIndexGetMaximumTermID (page 2756)
Gets the highest-numbered term ID in an index.

SKIndexDocumentIteratorCreate (page 2749)
Creates an index-based iterator for document URL objects owned by a parent document URL object.

SKIndexDocumentIteratorCopyNext (page 2748)
Obtains the next document URL object from an index using a document iterator.

SKIndexGetAnalysisProperties (page 2751)
Gets the text analysis properties of an index.

SKIndexMoveDocument (page 2758)
Changes the parent of a document URL object in an index.

`SKIndexRemoveDocument` (page 2761)
Removes a document URL object and its children, if any, from an index.

`SKIndexRenameDocument` (page 2761)
Changes the name of a document URL object in an index.

`SKIndexSetMaximumBytesBeforeFlush` (page 2763)
Not recommended. Sets the memory size limit for updates to an index, measured in bytes.

`SKIndexGetMaximumBytesBeforeFlush` (page 2755)
Not recommended. Gets the memory size limit for updates to an index, measured in bytes.

`SKIndexDocumentIteratorGetTypeID` (page 2750) Deprecated in Mac OS X v10.5
Gets the type identifier for Search Kit document iterators.

Working With Text Importers

Search Kit can import the textual content of file-based documents into indexes using the Spotlight metadata importers.

`SKLoadDefaultExtractorPlugIns` (page 2763)
Tells Search Kit to use the Spotlight metadata importers.

Working with Documents and Terms

From Search Kit's perspective, a document is anything that contains text—an RTF document, a PDF file, a Mail message, an Address Book entry, an Internet URL, the result of a database query, and so on.

The functions in this section let your application create new document URL objects (SKDocumentRefs), retrieve metadata from documents, get information on document hierarchies, and work with documents and their terms in the context of Search Kit indexes.

`SKDocumentCreateWithURL` (page 2735)
Creates a document URL object from a CFURL object.

`SKDocumentCreate` (page 2734)
Creates a document URL object based on a scheme, parent, and name.

`SKDocumentCopyURL` (page 2734)
Builds a CFURL object from a document URL object.

`SKDocumentGetName` (page 2736)
Gets the name of a document URL object.

`SKDocumentGetParent` (page 2736)
Gets a document URL object's parent.

`SKDocumentGetSchemeName` (page 2737)
Gets the scheme name for a document URL object.

`SKDocumentGetTypeID` (page 2737)
Gets the type identifier for Search Kit document URL objects.

`SKIndexCopyDocumentForDocumentID` (page 2741)
Obtains a document URL object from an index.

`SKIndexCopyInfoForDocumentIDs` (page 2744)
Gets document names and parent IDs based on document IDs.

SKIndexCopyDocumentRefsForDocumentIDs (page 2743)
Gets document URL objects based on document IDs.

SKIndexCopyDocumentURLsForDocumentIDs (page 2744)
Gets document URLs based on document IDs.

SKIndexCopyDocumentIDArrayForTermID (page 2742)
Obtains document IDs for documents that contain a given term.

SKIndexCopyTermIDArrayForDocumentID (page 2745)
Obtains the IDs for the terms of an indexed document.

SKIndexCopyTermStringForTermID (page 2746)
Obtains a term, specified by ID, from an index.

SKIndexGetTermIDForTermString (page 2757)
Gets the ID for a term in an index.

SKIndexSetDocumentProperties (page 2762)
Sets the application-defined properties of a document URL object.

SKIndexCopyDocumentProperties (page 2742)
Obtains the application-defined properties of an indexed document.

SKIndexGetDocumentState (page 2753)
Gets the current indexing state of a document URL object in an index.

SKIndexGetDocumentTermCount (page 2753)
Gets the number of terms for a document in an index.

SKIndexGetDocumentTermFrequency (page 2754)
Gets the number of occurrences of a term in a document.

SKIndexGetTermDocumentCount (page 2756)
Gets the number of documents containing a given term represented in an index.

SKIndexGetDocumentID (page 2752)
Gets the ID of a document URL object in an index.

Fast Asynchronous Searching

In Mac OS X v10.4 and later, Search Kit's fast asynchronous searching replaces synchronous searching. Synchronous searching, which relied on search groups, is deprecated.

SKSearchCreate (page 2764)
Creates an asynchronous search object for querying an index, and initiates search.

SKSearchFindMatches (page 2766)
Extracts search result information from a search object.

SKSearchCancel (page 2764)
Cancels an asynchronous search request.

SKSearchGetTypeID (page 2768)
Gets the type identifier for Search Kit search objects.

Working With Summarization

Search Kit's Summarization functions supplant those in Apple's Find by Content API.

`SKSummaryCreateWithString` (page 2777)
Creates a summary object based on a text string.

`SKSummaryGetSentenceSummaryInfo` (page 2779)
Gets detailed information about a body of text for constructing a custom sentence-based summary string.

`SKSummaryGetParagraphSummaryInfo` (page 2777)
Gets detailed information about a body of text for constructing a custom paragraph-based summary string.

`SKSummaryGetSentenceCount` (page 2778)
Gets the number of sentences in a summarization object.

`SKSummaryGetParagraphCount` (page 2777)
Gets the number of paragraphs in a summarization object.

`SKSummaryCopySentenceAtIndex` (page 2776)
Gets a specified sentence from the text in a summarization object.

`SKSummaryCopyParagraphAtIndex` (page 2775)
Gets a specified paragraph from the text in a summarization object.

`SKSummaryCopySentenceSummaryString` (page 2776)
Gets a text string consisting of a summary with, at most, the requested number of sentences.

`SKSummaryCopyParagraphSummaryString` (page 2775)
Gets a text string consisting of a summary with, at most, the requested number of paragraphs.

`SKSummaryGetTypeID` (page 2779)
Gets the type identifier for Search Kit summarization objects.

Legacy Support for Synchronous Searching

Developers should avoid using the functions listed in this section; instead, use the replacement functions that are recommended. Search Kit retains the functions in this section for backward compatibility.

`SKSearchGroupGetTypeID` (page 2769)
Deprecated. Use asynchronous searching with SKSearchCreate instead, which does not employ search groups.

`SKSearchResultsGetTypeID` (page 2774)
Gets the type identifier for Search Kit search results. (Deprecated. Use `SKSearchCreate` (page 2764) instead.)

`SKSearchGroupCopyIndexes` (page 2768) Deprecated in Mac OS X v10.4
Obtains the indexes for a search group. (Deprecated. Use asynchronous searching with SKSearchCreate instead, which does not employ search groups.)

`SKSearchGroupCreate` (page 2769) Deprecated in Mac OS X v10.4
Creates a search group as an array of references to indexes. (Deprecated. Use asynchronous searching with SKSearchCreate instead, which does not employ search groups.)

`SKSearchResultsCopyMatchingTerms` (page 2770) Deprecated in Mac OS X v10.4
Obtains the terms in a document that match a query. (Deprecated. Use `SKSearchCreate` (page 2764) instead.)

`SKSearchResultsCreateWithDocuments` (page 2770) Deprecated in Mac OS X v10.4
Finds documents similar to given example documents. (Deprecated. Use `SKSearchCreate` (page 2764) instead.)

`SKSearchResultsCreateWithQuery` (page 2772) Deprecated in Mac OS X v10.4
Queries the indexes in a search group. (Deprecated. Use `SKSearchCreate` (page 2764) instead.)

`SKSearchResultsGetCount` (page 2773) Deprecated in Mac OS X v10.4
Gets the total number of found items in a search. (Deprecated. Use `SKSearchCreate` (page 2764) instead.)

`SKSearchResultsGetInfoInRange` (page 2773) Deprecated in Mac OS X v10.4
Extracts information from a Search Kit query result. (Deprecated. Use `SKSearchCreate` (page 2764) instead.)

Functions

SKDocumentCopyURL

Builds a CFURL object from a document URL object.

```
CFURLRef SKDocumentCopyURL (
    SKDocumentRef   inDocument
);
```

Parameters

inDocument
The document URL object (SKDocumentRef) that you want a `CFURLRef` object for.

Return Value

A `CFURLRef` object representing a document location, or `NULL` on failure.

Discussion

You can use this function to create a *CFURL Reference* object to represent a document's location. Do this to gain access to the Core Foundation functionality provided by CFURL. This functionality includes accessing parts of the URL string, getting properties of the URL, and converting the URL to other representations.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`SKDocument.h`

SKDocumentCreate

Creates a document URL object based on a scheme, parent, and name.

```
SKDocumentRef SKDocumentCreate (
CFStringRef     inScheme,
SKDocumentRef   inParent,
CFStringRef     inName
);
```

Parameters

inScheme

The scheme to use—analogous to the scheme of a URL. Only documents referenced with the "file" scheme can be read by the SKIndexAddDocument (page 2738) function. The scheme can be anything you like if you use the SKIndexAddDocumentWithText (page 2739) function. The scheme can be NULL, in which case it will be set to be the same scheme as the document URL object's (SKDocumentRef's) parent. For more information on schemes, see http://www.iana.org/assignments/uri-schemes.html.

inParent

The document URL object one step up in the document hierarchy. Can be NULL.

inName

The name of the document that you're creating a document URL object for. For the "file" scheme, it is the name of the file or the container, not its path. The path can be constructed by following parent links. The maximum length for a document name is 256 bytes.

Return Value

The new document URL object, or NULL on failure.

Discussion

The new document URL object's (SKDocumentRef's) parent can be NULL, but you must specify either a scheme or a parent. When your application no longer needs the document URL object, dispose of it by calling CFRelease.

Availability

Available in Mac OS X v10.3 and later.

Declared In

SKDocument.h

SKDocumentCreateWithURL

Creates a document URL object from a CFURL object.

```
SKDocumentRef SKDocumentCreateWithURL (
    CFURLRef    inURL
);
```

Parameters

inURL

The URL for the document URL object (SKDocumentRef) you are creating. The scheme of the document URL object gets set to the scheme of the URL used. Only URLs with a scheme of "file" can be used with the SKIndexAddDocument (page 2738) function, but the URL scheme may be anything you like if you use the SKIndexAddDocumentWithText (page 2739) function. For more information on schemes, see http://www.iana.org/assignments/uri-schemes.html.

Return Value

The new document URL object (SKDocumentRef), or NULL if the document URL object could not be created.

Discussion
Use `SKDocumentCreateWithURL` to create a unique reference to a file or to another, arbitrary URL that your application will use as a document URL object (SKDocumentRef). When your application no longer needs the document URL object, dispose of it by calling `CFRelease`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKDocument.h`

SKDocumentGetName

Gets the name of a document URL object.

```
CFStringRef SKDocumentGetName (
    SKDocumentRef   inDocument
);
```

Parameters
`inDocument`
The document URL object (SKDocumentRef) whose name you want to get.

Return Value
A `CFStringRef` object containing the document URL object's name, or `NULL` on failure.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKDocument.h`

SKDocumentGetParent

Gets a document URL object's parent.

```
SKDocumentRef SKDocumentGetParent (
    SKDocumentRef   inDocument
);
```

Parameters
`inDocument`
The document URL object (SKDocumentRef) whose parent you want to get.

Return Value
The parent document URL object, or `NULL` on failure.

Discussion
As described in `SKDocumentRef` (page 2781), Search Kit manages document locations in terms of URLs as Document URL objects (SKDocumentRefs). The parent document URL object typically contains the document's URL up to but not including the document name.

Typically, document URL objects contain the complete URL to a file-based document. But you can use this function iteratively to build up the complete file-system path for a document that you are managing as part of a document hierarchy. See `SKDocumentRef` (page 2781) for more on this.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKDocument.h`

SKDocumentGetSchemeName

Gets the scheme name for a document URL object.

```
CFStringRef SKDocumentGetSchemeName (
    SKDocumentRef   inDocument
);
```

Parameters

`inDocument`
The document URL object (SKDocumentRef) whose scheme you want to get.

Return Value
A `CFStringRef` object containing the document URL object's scheme name, or `NULL` on failure.

Discussion
The scheme of a document URL object (SKDocumentRef), which represents how it can be accessed, can be any character string but is typically "`file`" or "`http`". The scheme is one of a Search Kit document URL object's three properties—see `SKDocumentRef` (page 2781) for details.

For more information on schemes, see http://www.iana.org/assignments/uri-schemes.html

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKDocument.h`

SKDocumentGetTypeID

Gets the type identifier for Search Kit document URL objects.

```
CFTypeID SKDocumentGetTypeID (void);
```

Return Value
A `CFTypeID` object containing the type identifier for the document URL object (SKDocumentRef).

Discussion
Search Kit represents document URL objects with the `SKDocumentRef` (page 2781) opaque type. If your code needs to determine whether a particular data type is a document URL object, you can use this function along with the `CFGetTypeID` function and perform a comparison.

Never hard-code the document URL object type ID because it can change from one release of Mac OS X to another.

Availability
Available in Mac OS X v10.3 and later.

Declared In
SKDocument.h

SKIndexAddDocument

Adds location information for a file-based document, and the document's textual content, to an index.

```
Boolean SKIndexAddDocument (
    SKIndexRef      inIndex,
    SKDocumentRef   inDocument,
    CFStringRef     inMIMETypeHint,
    Boolean         inCanReplace
);
```

Parameters

inIndex

The index you are adding the document URL object to.

inDocument

The document URL object (SKDocumentRef), containing a file-based document's location information, to add to the index. You can release the document URL object immediately after adding it to the index.

inMIMETypeHint

The MIME type hint for the specified file-based document. Can be NULL. In Search Kit, common MIME type hints include text/plain, text/rtf, text/html, text/pdf, and application/msword.

Specify a MIME type hint to help Spotlight determine which of its metadata importers to use when Search Kit is indexing a file-based document. Search Kit uses filename extensions and type/creator codes in attempting to determine file types when indexing files. See SKLoadDefaultExtractorPlugIns (page 2763). You can circumvent Search Kit's file type determination process, or override it, by using a MIME type hint.

inCanReplace

A Boolean value specifying whether Search Kit will overwrite a document's index entry (true, indicated by 1 or kCFBooleanTrue), or retain the entry if it exists (false, indicated by 0 or kCFBoolenFalse).

Return Value
A Boolean value of true on success, or false on failure. Also returns false if the document has an entry in the index and *inCanReplace* is set to false.

Discussion
The document scheme must be of type "file" to use this function. If it's not, call SKIndexAddDocumentWithText (page 2739) instead. For more information on schemes, see http://www.iana.org/assignments/uri-schemes.html.

This function uses the referenced document and the optional MIME type hint to get the document's textual content using the Spotlight metadata importers. If you do not supply a MIME type hint, Spotlight's importers will use filename extensions and type/creator codes to guess file types.

Search Kit indexes any nonexecutable file associated with a document URL object (SKDocumentRef) that you hand to this function, even nontext files such as images. Your application takes responsibility for ensuring that the document URL objects you pass to SKIndexAddDocument are in fact the locations of files you want to index.

If your application did not call SKLoadDefaultExtractorPlugIns (page 2763), Search Kit indexes the first 10 MB of a document. Otherwise, Search Kit indexes the entire document up to the index file size limit of 4 GB.

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

A single Search Kit index can hold up to 4 billion document URL objects and their associated textual content.

Special Considerations

In the current implementation of Search Kit, some functions do not provide expected results unless you follow SKIndexAddDocument with a call to SKIndexFlush (page 2750). The affected functions include SKIndexGetDocumentCount (page 2751), SKIndexGetDocumentTermCount (page 2753), SKIndexGetDocumentTermFrequency (page 2754), and SKIndexGetTermDocumentCount (page 2756). However, in typical use this won't be an issue, because applications call these functions after a search, and you must call SKIndexFlush before a search.

Version Notes

In versions of Mac OS X prior to Mac OS X v10.4, Search Kit used its own text extractor plug-ins rather than using the Spotlight metadata importers. See SKLoadDefaultExtractorPlugIns (page 2763) and http://developer.apple.com/macosx/tiger/spotlight.html.

Availability

Available in Mac OS X v10.3 and later.

Declared In

SKIndex.h

SKIndexAddDocumentWithText

Adds a document URL object, and the associated document's textual content, to an index.

```
Boolean SKIndexAddDocumentWithText (
    SKIndexRef      inIndex,
    SKDocumentRef   inDocument,
    CFStringRef     inDocumentText,
    Boolean         inCanReplace
);
```

Parameters

inIndex

The index to which you are adding the document URL object (SKDocumentRef).

inDocument

The document URL object to add.

inDocumentText

The document text. Can be NULL.

inCanReplace

A Boolean value specifying whether Search Kit will overwrite a document's index entry (true, indicated by 1 or kCFBooleanTrue), or retain the entry if it exists (false, indicated by 0 or kCFBoolenFalse).

Return Value

A Boolean value of true on success, or false on failure. Also returns false if the document has an entry in the index and *inCanReplace* is set to false.

Discussion
Use this function to add the textual contents of arbitrary document types to an index. With this function, your application takes responsibility for getting textual content and handing it to the index as a CFString object. Because of this, your application can define what it considers to be a document—a database record, a tagged field in an XML document, an object in memory, a text file, and so on.

Search Kit will index any size text string that you give it, up to its 4 GB index file size limit.

To add the textual content of file-based documents to a Search Kit index, you can use this function or take advantage of Search Kit's ability to locate and read certain on-disk, file-based document types—see `SKIndexAddDocument` (page 2738).

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

A single Search Kit index file can be up to 4 GB in size.

Special Considerations

In Mac OS X v10.3, some functions do not provide expected results unless you follow a call to `SKIndexAddDocumentWithText` with a call to `SKIndexFlush` (page 2750). The affected functions include `SKIndexGetDocumentCount` (page 2751), `SKIndexGetDocumentTermCount` (page 2753), `SKIndexGetDocumentTermFrequency` (page 2754), and `SKIndexGetTermDocumentCount` (page 2756). However, in typical use this won't be an issue, because applications call these functions after a search, and you must call `SKIndexFlush` before a search.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexClose

Closes an index.

```
void SKIndexClose (
    SKIndexRef      inIndex
);
```

Parameters

inIndex
The index to close.

Discussion
When your application no longer needs an index that it has opened or created, call `SKIndexClose`. Calling this function is equivalent to calling `CFRelease` on an index.

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`SKIndex.h`

SKIndexCompact

Invokes all pending updates associated with an index, compacts the index if compaction is needed, and commits all changes to backing store.

```
Boolean SKIndexCompact (
    SKIndexRef  inIndex
);
```

Parameters

inIndex

The index you want to compact.

Return Value

A Boolean value of `true` on success, or `false` on failure.

Discussion

Over time, as document URL objects (SKDocumentRefs) and associated contents get added to and removed from an index, the index's disk or memory footprint may grow due to fragmentation.

Compacting can take a significant amount of time. Do not call `SKIndexCompact` on the main thread in an application with a user interface. Call it only if the index is significantly fragmented and according to the needs of your application.

Calling `SKIndexCompact` changes the block allocation for an index's backing store. Close all clients of an index before calling this function.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`SKIndex.h`

SKIndexCopyDocumentForDocumentID

Obtains a document URL object from an index.

```
SKDocumentRef SKIndexCopyDocumentForDocumentID (
    SKIndexRef      inIndex,
    SKDocumentID    inDocumentID
);
```

Parameters

inIndex

The index containing the document URL object (SKDocumentRef).

inDocumentID

The ID of the document URL object you want to copy.

Return Value

A Search Kit document URL object.

Version Notes

In versions of Mac OS X prior to Mac OS X v10.4, the parameter type for *inDocumentID* was `CFIndex`. The parameter type in Mac OS X v10.4 and later is `SKDocumentID`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
SKIndex.h

SKIndexCopyDocumentIDArrayForTermID

Obtains document IDs for documents that contain a given term.

```
CFArrayRef SKIndexCopyDocumentIDArrayForTermID (
    SKIndexRef      inIndex,
    CFIndex         inTermID
);
```

Parameters

inIndex

The index to search.

inTermID

The ID of the term to search for.

Return Value
An array of CFNumbers, each the ID for a document URL object that points to a document containing the search term.

Discussion
SKIndexCopyDocumentIDArrayForTermID searches a single index for documents that contain a given term. The search uses a term ID, not a term string. To get the ID of a term, use SKIndexGetTermIDForTermString (page 2757).

Term IDs are index-specific; that is, a term has a different ID in each index in which it appears. If you want to search for all the documents containing a term in a set of indexes, call this function in turn for each index, using the index-specific term ID in each case.

Availability
Available in Mac OS X v10.3 and later.

Declared In
SKIndex.h

SKIndexCopyDocumentProperties

Obtains the application-defined properties of an indexed document.

```
CFDictionaryRef SKIndexCopyDocumentProperties (
    SKIndexRef      inIndex,
    SKDocumentRef   inDocument
);
```

Parameters

inIndex

The index containing the document URL object (SKDocumentRef) whose properties you want to copy.

inDocument
The document URL object whose properties you want to copy.

Return Value
A CFDictionary object containing the document URL object's (SKDocumentRef's) properties, or `NULL` on failure.

Discussion
Search Kit document URL objects (SKDocumentRefs) can have an optional, application-defined properties dictionary to hold any information you'd like to associate with the document represented by a document URL object—such as timestamp, keywords, and so on. Use `SKIndexSetDocumentProperties` (page 2762) to add a properties dictionary to a document URL object, and this function to obtain a copy of the dictionary.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexCopyDocumentRefsForDocumentIDs

Gets document URL objects based on document IDs.

```
void SKIndexCopyDocumentRefsForDocumentIDs (
    SKIndexRef      inIndex,
    CFIndex         inCount,
    SKDocumentID    *inDocumentIDsArray,
    SKDocumentRef   *outDocumentRefsArray
);
```

Parameters

inIndex
The index containing the document information.

inCount
The number of document IDs in *inDocumentIDsArray*.

inDocumentIDsArray
Points to an array of document IDs corresponding to the document URL objects (SKDocumentRefs) you want.

outDocumentRefsArray
On input, a pointer to an array for document URL objects. On output, points to the previously allocated array, which now contains document URL objects corresponding to the document IDs in `inDocumentIDsArray`.

When finished with the document URL objects array, dispose of it by calling `CFRelease` on each array element.

Discussion
The `SKIndexCopyDocumentRefsForDocumentIDs` function lets you get a batch of document URL objects (SKDocumentRef objects) in one step, based on a list of document IDs.

If you want to get lightweight URLs in the form of CFURL objects instead, use `SKIndexCopyDocumentURLsForDocumentIDs` (page 2744).

Availability
Available in Mac OS X v10.4 and later.

Declared In
SKSearch.h

SKIndexCopyDocumentURLsForDocumentIDs

Gets document URLs based on document IDs.

```
void SKIndexCopyDocumentURLsForDocumentIDs (
    SKIndexRef      inIndex,
    CFIndex         inCount,
    SKDocumentID    *inDocumentIDsArray,
    CFURLRef        *outDocumentURLsArray
);
```

Parameters

inIndex
The index containing the document information.

inCount
The number of document IDs in *inDocumentIDsArray*.

inDocumentIDsArray
Points to an array of document IDs corresponding to the document URLs (CFURL objects) you want.

outDocumentURLsArray
On input, a pointer to an array for document URLs (CFURL objects). On output, points to the previously allocated array, which now contains document URLs corresponding to the document IDs in `inDocumentIDArray`.

When finished with the document URL array, dispose of it by calling `CFRelease` on each array element.

Discussion
The `SKIndexCopyDocumentURLsForDocumentIDs` function lets you get a batch of document URLs (CFURL objects) in one step, based on a list of document IDs.

If you want to get Search Kit Document URL objects (SKDocumentRefs) instead, use `SKIndexCopyDocumentRefsForDocumentIDs` (page 2743).

Availability
Available in Mac OS X v10.4 and later.

Declared In
SKSearch.h

SKIndexCopyInfoForDocumentIDs

Gets document names and parent IDs based on document IDs.

```
void SKIndexCopyInfoForDocumentIDs (
    SKIndexRef      inIndex,
    CFIndex         inCount,
    SKDocumentID    *inDocumentIDsArray,
    CFStringRef     *outNamesArray,
    SKDocumentID    *outParentIDsArray
);
```

Parameters

inIndex

The index containing the document information.

inCount

The number of document IDs in *inDocumentIDsArray*.

inDocumentIDsArray

Points to an array of document IDs representing the documents whose names and parent IDs you want.

outNamesArray

On input, a pointer to an array for document names. On output, points to the previously allocated array, which now contains the document names corresponding to the document IDs in *inDocumentIDsArray*. May be `NULL` on input if you don't want to get the document names.

When finished with the names array, dispose of it by calling `CFRelease` on each array element.

outParentIDsArray

On input, a pointer to an array for parent document IDs. On output, points to the previously allocated array, which now contains document IDs representing the parents of the documents whose IDs are in *inDocumentIDsArray*. May be `NULL` on input if you don't want to get the parent document IDs.

Discussion

The `SKIndexCopyInfoForDocumentIDs` function lets you get a batch of document names and parent document IDs in one step, based on a list of document IDs.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`SKSearch.h`

SKIndexCopyTermIDArrayForDocumentID

Obtains the IDs for the terms of an indexed document.

```
CFArrayRef SKIndexCopyTermIDArrayForDocumentID (
    SKIndexRef      inIndex,
    SKDocumentID    inDocumentID
);
```

Parameters

inIndex

The index containing the document URL object (SKDocumentRef) and associated textual content.

inDocumentID

The ID of the document whose term IDs you are copying.

Return Value
A CFArray containing CFNumbers, each of which represents the ID for a term in a document.

Discussion
To derive the list of terms contained in a document, use this function to obtain an array of the term IDs, then convert each ID into the corresponding term with the `SKIndexCopyTermStringForTermID` (page 2746) function.

Version Notes
In versions of Mac OS X prior to Mac OS X v10.4, the parameter type for `inDocumentID` was `CFIndex`. In Mac OS X v10.4 and later, the parameter type is `SKDocumentID`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexCopyTermStringForTermID

Obtains a term, specified by ID, from an index.

```
CFStringRef SKIndexCopyTermStringForTermID (
    SKIndexRef      inIndex,
    CFIndex         inTermID
);
```

Parameters

`inIndex`
The index whose terms you are searching.

`inTermID`
The ID of the term whose string you want.

Return Value
A CFString containing the term specified by `inTermID`.

Discussion
When your application has the ID of a term, perhaps as a result of calling `SKIndexCopyTermIDArrayForDocumentID` (page 2745), use this function to derive the term's text string.

To perform the inverse operation of deriving a term ID from a term string in a given index, use `SKIndexGetTermIDForTermString` (page 2757).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexCreateWithMutableData

Creates a named index stored in a CFMutableData object.

```
SKIndexRef SKIndexCreateWithMutableData (
    CFMutableDataRef    inData,
    CFStringRef         inIndexName,
    SKIndexType         inIndexType,
    CFDictionaryRef     inAnalysisProperties
);
```

Parameters

inData

An empty CFMutableData object to contain the index being created.

inIndexName

The name of the index. If you call this function with `inIndexName` set to `NULL`, Search Kit assigns the index the default index name `IADefaultIndex`. If you then attempt to create a second index in the same file without assigning a name, no second index is created and this function returns `NULL`. Search Kit does not currently support retrieving index names from an index.

inIndexType

The index type. See "`SKIndexType`" (page 2788).

inAnalysisProperties

The text analysis properties dictionary, which optionally sets the minimum term length, stopwords, term substitutions, maximum unique terms to index, and proximity support (for phrase-based searches) when creating the index. See "`Text Analysis Keys`" (page 2784). The `inAnalysisProperties` parameter can be `NULL`, in which case Search Kit applies the default dictionary, which is `NULL`.

Return Value
The newly created index.

Discussion
`SKIndexCreateWithMutableData` creates an index in memory as a CFMutableData object. Search Kit indexes are initially empty. A memory-based index is useful for quick searching and when your application doesn't need persistent storage. To create a disk-based, persistent index, use `SKIndexCreateWithURL` (page 2747).

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

When your application no longer needs the index, dispose of it by calling `SKIndexClose` (page 2740).

Special Considerations

You cannot use `CFMakeCollectable` with SKIndex objects.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexCreateWithURL

Creates a named index in a file whose location is specified with a CFURL object.

```
SKIndexRef SKIndexCreateWithURL (
    CFURLRef            inURL,
    CFStringRef         inIndexName,
    SKIndexType         inIndexType,
    CFDictionaryRef     inAnalysisProperties
);
```

Parameters

inURL

The location of the index.

inIndexName

The name of the index. If you call this function with `inIndexName` set to `NULL`, Search Kit assigns the index the default index name `IADefaultIndex`. If you then attempt to create a second index in the same file without assigning a name, no second index is created and this function returns `NULL`. Search Kit does not currently support retrieving index names from an index.

inIndexType

The index type. See "SKIndexType" (page 2788).

inAnalysisProperties

The text analysis properties dictionary, which optionally sets the minimum term length, stopwords, term substitutions, maximum unique terms to index, and proximity support (for phrase-based searches) when creating the index. See "Text Analysis Keys" (page 2784). To get the analysis properties of an index, use the `SKIndexGetAnalysisProperties` (page 2751) function. The `inAnalysisProperties` parameter can be `NULL`, in which case Search Kit applies the default dictionary, which is `NULL`.

Return Value

A unique reference to the newly created index.

Discussion

`SKIndexCreateWithURL` creates an index in a file. Search Kit indexes are initially empty. Use this function when your application needs persistent storage of an index. To create a memory-based, nonpersistent index, use `SKIndexCreateWithMutableData` (page 2746).

A file can contain more than one index. To add a new index to an existing file, use the same value for `inURL` and supply a new name for `inIndexName`.

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

When your application no longer needs the index, dispose of it by calling `SKIndexClose` (page 2740).

Special Considerations

You cannot use `CFMakeCollectable` with SKIndex objects.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`SKIndex.h`

SKIndexDocumentIteratorCopyNext

Obtains the next document URL object from an index using a document iterator.

```
SKDocumentRef SKIndexDocumentIteratorCopyNext (
    SKIndexDocumentIteratorRef  inIterator
);
```

Parameters

inIterator

The index-based document iterator. See `SKIndexDocumentIteratorCreate` (page 2749) for information on creating an document iterator, and `SKIndexDocumentIteratorRef` (page 2781) for more about iterators.

Return Value
The next document URL object (SKDocumentRef) in the index.

Discussion
This function returns `NULL` when there are no more document URL objects (SKDocumentRefs) in the index. When finished iterating, your application must call `CFRelease` on all retrieved document URL objects that are non-`NULL`.

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexDocumentIteratorCreate

Creates an index-based iterator for document URL objects owned by a parent document URL object.

```
SKIndexDocumentIteratorRef SKIndexDocumentIteratorCreate (
    SKIndexRef      inIndex,
    SKDocumentRef   inParentDocument
);
```

Parameters

inIndex

The index you want to iterate across.

inParentDocument

The document URL object (SKDocumentRef) that is the parent of the document URL objects you want to examine. Pass `NULL` to get the top item in an index. See `SKDocumentRef` (page 2781) for a discussion of how to get the full URL for a document URL object.

Return Value
An index-based document iterator.

Discussion
When you want to iterate across all the documents represented in an index, use this function to create an iterator and then call `SKIndexDocumentIteratorCopyNext` (page 2748) in turn for each document URL object (SKDocumentRef) in the index.

Document iterators iterate over a single level of an index. Your code is responsible for descending through a hierarchy of documents in an index.

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

When your application no longer needs the iterator, dispose of it by calling `CFRelease`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexDocumentIteratorGetTypeID

Gets the type identifier for Search Kit document iterators.

```
CFTypeID SKIndexDocumentIteratorGetTypeID (void);
```

Return Value
A CFTypeID object containing the type identifier for the SKIndexDocumentIterator opaque type.

Discussion
Search Kit represents document iterators with the `SKIndexDocumentIteratorRef` (page 2781) opaque type. If your code needs to determine whether a particular data type is a document iterator, you can use this function along with the `CFGetTypeID` function and perform a comparison.

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

Never hard-code the document iterator type ID because it can change from one release of Mac OS X to another.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexFlush

Invokes all pending updates associated with an index and commits them to backing store.

```
Boolean SKIndexFlush (
    SKIndexRef  inIndex
);
```

Parameters

inIndex
The index you want to update and commit to backing store.

Return Value
A Boolean value of `true` on success, or `false` on failure.

Discussion
An on-disk or memory-based index becomes stale when your application updates it by adding or removing a document entry. A search on an index in such a state won't have access to the nonflushed updates. The solution is to call this function before searching. `SKIndexFlush` flushes index-update information and commits memory-based index caches to disk, in the case of an on-disk index, or to a memory object, in the case of a memory-based index. In both cases, calling this function makes the state of the index consistent.

Before searching an index, always call `SKIndexFlush`, even though the flush process may take up to several seconds. If there are no updates to commit, a call to `SKIndexFlush` does nothing and takes minimal time.

A new Search Kit index does not have term IDs until it is flushed.

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexGetAnalysisProperties

Gets the text analysis properties of an index.

```
CFDictionaryRef SKIndexGetAnalysisProperties (
    SKIndexRef      inIndex
);
```

Parameters
inIndex
The index whose text-analysis properties you want to get.

Return Value
A CFDictionary object containing the index's text-analysis properties. On failure, returns `NULL`.

Discussion
The text analysis properties of an index determine how searches behave when querying the index. You set the analysis properties when creating an index with the `SKIndexCreateWithURL` (page 2747) or `SKIndexCreateWithMutableData` (page 2746) functions. For more information on text-analysis properties, see "`Text Analysis Keys`" (page 2784).

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexGetDocumentCount

Gets the total number of documents represented in an index.

```
CFIndex SKIndexGetDocumentCount (
    SKIndexRef      inIndex
);
```

Parameters

inIndex

The index whose document URL objects (SKDocumentRefs) you want to count.

Return Value
A CFIndex object containing the number of document URL objects in the index. On failure, returns 0.

Discussion
Document URL objects (SKDocumentRefs) added to an index have an indexing state of `kSKDocumentStateIndexed`. See the "`SKDocumentIndexState`" (page 2786) enumeration.

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

Special Considerations

In the current implementation of Search Kit, `SKIndexGetDocumentCount` returns the number of documents represented in the on-disk index. If your application has added document URL objects to the index but has not yet called `SKIndexFlush` (page 2750), the document count may not be correct.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexGetDocumentID

Gets the ID of a document URL object in an index.

```
SKDocumentID SKIndexGetDocumentID (
    SKIndexRef      inIndex,
    SKDocumentRef   inDocument
);
```

Parameters

inIndex

The index containing the text of the document whose document URL object (SKDocumentRef) ID you want.

inDocument

The document URL object whose ID you want.

Return Value
A document ID object.

Discussion
The document ID identifies a document URL object (SKDocumentRef) in an index. The ID is available as soon as you add a document URL object to an index using `SKIndexAddDocumentWithText` (page 2739) or `SKIndexAddDocument` (page 2738).

Availability
Available in Mac OS X v10.3 and later.

Declared In
SKIndex.h

SKIndexGetDocumentState

Gets the current indexing state of a document URL object in an index.

```
SKDocumentIndexState SKIndexGetDocumentState (
    SKIndexRef      inIndex,
    SKDocumentRef   inDocument
);
```

Parameters

inIndex

The index containing the document URL object (SKDocumentRef) whose indexing state you want.

inDocument

The document URL object whose indexing state you want.

Return Value
A value indicating the document URL object's indexing state.

Discussion
A document URL object (SKDocumentRef) can be in one of four states, as defined by the "SKDocumentIndexState" (page 2786) enumeration: not indexed, indexed, not in the index but will be added after the index is flushed or closed, and in the index but will be deleted after the index is flushed or closed.

Availability
Available in Mac OS X v10.3 and later.

Declared In
SKIndex.h

SKIndexGetDocumentTermCount

Gets the number of terms for a document in an index.

```
CFIndex SKIndexGetDocumentTermCount (
    SKIndexRef      inIndex,
    SKDocumentID    inDocumentID
);
```

Parameters

inIndex

The index containing the text of the document whose term count you want.

inDocumentID

The ID of the document URL object (SKDocumentRef) whose term count you want. Obtain a document ID by calling SKIndexGetDocumentID (page 2752).

Return Value
A CFIndex object containing the number of terms in a document.

Version Notes
versions of Mac OS X prior to Mac OS X v10.4, the parameter type for *inDocumentID* was `CFIndex`. In Mac OS X v10.4 and later, the parameter type is `SKDocumentID`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexGetDocumentTermFrequency

Gets the number of occurrences of a term in a document.

```
CFIndex SKIndexGetDocumentTermFrequency (
    SKIndexRef      inIndex,
    SKDocumentID    inDocumentID,
    CFIndex         inTermID
);
```

Parameters

inIndex
The index containing the text of the document whose term count you are interested in.

inDocumentID
The ID of the document URL object whose associated term count you are interested in. Obtain a document ID by calling `SKIndexGetDocumentID` (page 2752).

inTermID
The ID of the term whose number of occurrences you want.

Return Value
A CFIndex object containing the number of occurrences of a term in a document.

Version Notes
In versions of Mac OS X prior to Mac OS X v10.4, the parameter type for `inDocumentID` was `CFIndex`. In Mac OS X v10.4 and later, the parameter type is `SKDocumentID`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexGetIndexType

Gets the category of an index.

```
SKIndexType SKIndexGetIndexType (
    SKIndexRef      inIndex
);
```

Parameters

inIndex
The index whose category you want to know.

Return Value
The category of the index. See the "SKIndexType" (page 2788) enumeration for a list of the various index categories. On failure, returns a value of kSKIndexUnknown.

Discussion
As described in "SKIndexType" (page 2788), Search Kit offers four categories of index, each optimized for one or more types of searching.

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

Availability
Available in Mac OS X v10.3 and later.

Declared In
SKIndex.h

SKIndexGetMaximumBytesBeforeFlush

Not recommended. Gets the memory size limit for updates to an index, measured in bytes.

```
CFIndex SKIndexGetMaximumBytesBeforeFlush (
    SKIndexRef      inIndex
);
```

Special Considerations

This function is rarely needed and is likely to be deprecated. Apple recommends using the SKIndexFlush (page 2750) function along with the default memory size limit for index updates. Refer to the SKIndexSetMaximumBytesBeforeFlush function for more information.

Availability
Available in Mac OS X v10.3 and later.

Declared In
SKIndex.h

SKIndexGetMaximumDocumentID

Gets the highest-numbered document ID in an index.

```
SKDocumentID SKIndexGetMaximumDocumentID (
    SKIndexRef  inIndex
);
```

Parameters

inIndex
An index.

Return Value
A document ID object containing the highest-numbered document ID in the index.

Discussion
Use this function with SKIndexGetDocumentCount (page 2751) to determine whether an index is fragmented and in need of compaction. See SKIndexCompact (page 2741).

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

Version Notes
In versions of Mac OS X prior to Mac OS X v10.4, the return type for `SKIndexGetMaximumDocumentID` was `CFIndex`. The return type in Mac OS X v10.4 and later is `SKDocumentID`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexGetMaximumTermID

Gets the highest-numbered term ID in an index.

```
CFIndex SKIndexGetMaximumTermID (
    SKIndexRef  inIndex
);
```

Parameters

inIndex
An index.

Return Value
A CFIndex object containing the highest-numbered term ID in an index.

Discussion
A new Search Kit index does not have term IDs until it is flushed.

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexGetTermDocumentCount

Gets the number of documents containing a given term represented in an index.

```
CFIndex SKIndexGetTermDocumentCount (
    SKIndexRef      inIndex,
    CFIndex         inTermID
);
```

Parameters

inIndex
The index containing the text of the documents you want to examine.

inTermID
The terms whose occurrences you want to know.

Return Value
A CFIndex object containing the number of documents represented in an index that contain a given term.

Discussion
If you want to know in which documents a term appears across multiple indexes, call this function separately on each index. Before querying each index, get the index-specific term ID using `SKIndexGetTermIDForTermString` (page 2757).

To ensure that this function takes into account document URL objects (SKDocumentRefs) recently added to indexes, call `SKIndexFlush` (page 2750) on each index before calling this function.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexGetTermIDForTermString

Gets the ID for a term in an index.

```
CFIndex SKIndexGetTermIDForTermString (
    SKIndexRef      inIndex,
    CFStringRef     inTermString
);
```

Parameters

inIndex
The index you want to examine.

inTermString
The term string whose corresponding ID you want.

Return Value
A CFIndex object containing the term ID for a given term in an index. If the term isn't found, this function returns a value of `kCFNotFound`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexGetTypeID

Gets the type identifier for Search Kit indexes.

```
CFTypeID SKIndexGetTypeID (void);
```

Return Value
A CFTypeID object containing the type identifier for the SKIndex opaque type.

Discussion
Search Kit represents indexes with the `SKIndexRef` (page 2782) opaque type. If your code needs to determine whether a particular data type is an index, you can use this function along with the `CFGetTypeID` function and perform a comparison.

Never hard-code the index type ID because it can change from one release of Mac OS X to another.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexMoveDocument

Changes the parent of a document URL object in an index.

```
Boolean SKIndexMoveDocument (
    SKIndexRef      inIndex,
    SKDocumentRef   inDocument,
    SKDocumentRef   inNewParent
);
```

Parameters

inIndex
The index containing the document URL object (SKDocumentRef) you want to move.

inDocument
The document URL object you want to move.

inNewParent
The new parent document URL object for the document URL object you want to move.

Return Value
A Boolean value of `true` for a successful move, or `false` on failure.

Discussion
When your application moves a document, use this function to update the index to reflect the change.

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexOpenWithData

Opens an existing, named index for searching only.

```
SKIndexRef SKIndexOpenWithData (
    CFDataRef       inData,
    CFStringRef     inIndexName
);
```

Parameters

inData

The index to open.

inIndexName

The name of the index. Can be `NULL`, in which case this function attempts to open the index with the default name of `IADefaultIndex`.

Return Value

The named index, or `NULL` on failure.

Discussion

An index opened by `SKIndexOpenWithData` can be searched but not updated. To open an index for updating, use `SKIndexOpenWithMutableData` (page 2759).

If `inIndexName` is `NULL` and `inData` does not contain an index with the default name, this function returns `NULL`.

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

A call to `SKIndexOpenWithData` retains the opened index. When your application no longer needs the index, dispose of it by calling `SKIndexClose` (page 2740).

Special Considerations

You cannot use `CFMakeCollectable` with SKIndex objects.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`SKIndex.h`

SKIndexOpenWithMutableData

Opens an existing, named index for searching and updating.

```
SKIndexRef SKIndexOpenWithMutableData (
    CFMutableDataRef    inData,
    CFStringRef         inIndexName
);
```

Parameters

inData

The index to open.

inIndexName

The name of the index. Can be `NULL`, in which case this function attempts to open the index with the default name of `IADefaultIndex`.

Return Value
The named index, or `NULL` on failure.

Discussion
An index opened by `SKIndexOpenWithMutableData` may be searched or updated. To open an index for search only, use the `SKIndexOpenWithData` (page 2758) function.

If `inIndexName` is `NULL` and `inData` does not contain an index with the default name, this function returns `NULL`.

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

A call to `SKIndexOpenWithMutableData` retains the opened index. When your application no longer needs the index, dispose of it by calling `SKIndexClose` (page 2740).

Special Considerations

You cannot use `CFMakeCollectable` with SKIndex objects.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexOpenWithURL

Opens an existing, named index stored in a file whose location is specified with a CFURL object.

```
SKIndexRef SKIndexOpenWithURL (
    CFURLRef        inURL,
    CFStringRef     inIndexName,
    Boolean         inWriteAccess
);
```

Parameters

inURL

The location of the index.

inIndexName

The name of the index. Can be `NULL`.

inWriteAccess

A Boolean value indicating whether the index is open for updating. To open an index for searching only, pass `false` (`0` or `kCFBoolenFalse`). To open it for searching and updating, pass `true` (`1` or `kCFBooleanTrue`).

Return Value
The named index.

Discussion
A call to `SKIndexOpenWithURL` retains the opened index. When your application no longer needs the index, dispose of it by calling `SKIndexClose` (page 2740).

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

Special Considerations

You cannot use `CFMakeCollectable` with SKIndex objects.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexRemoveDocument

Removes a document URL object and its children, if any, from an index.

```
Boolean SKIndexRemoveDocument (
    SKIndexRef      inIndex,
    SKDocumentRef   inDocument
);
```

Parameters

inIndex

The index from which you want to remove the document URL object (SKDocumentRef).

inDocument

The document URL object to remove.

Return Value
A Boolean value of `true` on success, or `false` on failure.

Discussion
When your application deletes a document, use this function to update the index to reflect the change.

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexRenameDocument

Changes the name of a document URL object in an index.

```
Boolean SKIndexRenameDocument (
    SKIndexRef      inIndex,
    SKDocumentRef   inDocument,
    CFStringRef     inNewName
);
```

Parameters

inIndex

The index containing the document URL object (SKDocumentRef) whose name you want to change.

inDocument
The document URL object whose name you want to change.

inNewName
The new name for the document URL object.

Return Value
A Boolean value of `true` if the document URL object name was successfully changed, or `false` on failure.

Discussion
When your application changes the name of a document, use this function to update the index to reflect the change.

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexSetDocumentProperties

Sets the application-defined properties of a document URL object.

```
void SKIndexSetDocumentProperties (
    SKIndexRef          inIndex,
    SKDocumentRef       inDocument,
    CFDictionaryRef     inProperties
);
```

Parameters
inIndex
An index containing the document URL object (SKDocumentRef) whose properties you want to set.

inDocument
The document URL object whose properties you want to set.

inProperties
A CFDictionary object containing the properties to apply to the document URL object.

Discussion
Search Kit document URL objects (SKDocumentRefs) can have an optional, application-defined properties dictionary to hold any information you'd like to associate with the document represented by a document URL object—such as timestamp, keywords, and so on.

Use `SKIndexSetDocumentProperties` to persistently set application-defined properties for a document URL object in an index. This function replaces a document URL object's existing properties dictionary with the new one. To obtain a copy of a document URL object's properties dictionary, use `SKIndexCopyDocumentProperties` (page 2742).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexSetMaximumBytesBeforeFlush

Not recommended. Sets the memory size limit for updates to an index, measured in bytes.

```
void SKIndexSetMaximumBytesBeforeFlush (
    SKIndexRef  inIndex
    CFIndex     inBytesForUpdate
);
```

Discussion

This function is rarely needed and is likely to be deprecated. Search Kit keeps track of index updates that are not yet committed to disk. Apple recommends using the default memory size limit for index updates, which is currently 2 million bytes.

Special Considerations

Apple recommends use of the `SKIndexFlush` (page 2750) function instead of `SKIndexSetMaximumBytesBeforeFlush`.

Version Notes

In Mac OS X v10.3, the default memory size limit for index updates was 1 million bytes.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`SKIndex.h`

SKLoadDefaultExtractorPlugIns

Tells Search Kit to use the Spotlight metadata importers.

```
void SKLoadDefaultExtractorPlugIns (void);
```

Discussion

The Spotlight metadata importers determine the `kMDItemTextContent` property for each document passed to the `SKIndexAddDocument` (page 2738) function. See http://developer.apple.com/macosx/tiger/spotlight.html.

Call the `SKLoadDefaultExtractorPlugIns` function once at application launch to tell Search Kit to use the Spotlight metadata importers. The function `SKIndexAddDocument` (page 2738) will then use Spotlight's importers to extract the text from supported files and place that text into an index, leaving the markup behind.

Version Notes

In versions of Mac OS X prior to Mac OS X v10.4, Search Kit used its own set of default text extractor plug-ins. The file types supported by Search Kit's default text extractor plug-ins were:

- plaintext
- PDF
- HTML
- RTF
- Microsoft Word (.doc)

Availability
Available in Mac OS X v10.3 and later.

Declared In
SKIndex.h

SKSearchCancel

Cancels an asynchronous search request.

```
void SKSearchCancel (
    SKSearchRef                 inSearch
);
```

Parameters

inSearch

The search object whose associated asynchronous search you want to cancel.

Discussion
Call this function when you want to cancel an asynchronous search that you initiated with SKSearchCreate (page 2764). This function stops the search process if it is still in progress at the time. It does not dispose of the search object (SKSearchRef).

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

Availability
Available in Mac OS X v10.4 and later.

Declared In
SKSearch.h

SKSearchCreate

Creates an asynchronous search object for querying an index, and initiates search.

```
SKSearchRef SKSearchCreate (
    SKIndexRef              inIndex,
    CFStringRef             inQuery,
    SKSearchOptions         inSearchOptions
);
```

Parameters

inIndex

The index to query.

inQuery

The query string to search for.

inSearchOptions

The search options. May be NULL. See the "SKSearchOptions" (page 2787) enumeration for a description of the available options.

Return Value
A search object.

Discussion
This function creates an asynchronous search object for querying the document contents in an index. It also initiates the search on a separate thread.

After you create the search object, call `SKSearchFindMatches` (page 2766) to retrieve results. You can call `SKSearchFindMatches` immediately. To cancel a search, call `SKSearchCancel` (page 2764).

For normal (non-similarity-based) queries, Search Kit discerns the type of query—Boolean, prefix, phrase, and so on—from the syntax of the query itself. Moreover, Search Kit supports multiple query types within a single search. For example, the following query includes Boolean, prefix, and suffix searching:

```
appl* OR *ing
```

This query will return documents containing words that begin with "appl" as well as documents that contain words that end with "ing".

For similarity searches, specified with the `kSKSearchOptionFindSimilar` flag in the *`inSearchOptions`* parameter, `SKSearchCreate` ignores all query operators.

The query operators that `SKSearchCreate` recognizes for non-similarity searching are:

Table 45-1 Search Kit query operators for non-similarity searches

Operator	meaning
`AND`	Boolean `AND`
`&`	Boolean `AND`
`<space>`	Boolean `AND` by default when no other operator is present, or Boolean `OR` if specified by `kSKSearchOptionSpaceMeansOR`.
`OR`	Boolean inclusive `OR`
`|`	Boolean inclusive `OR`
`NOT`	Boolean `NOT` (see Special Considerations)
`!`	Boolean `NOT` (see Special Considerations)
*	Wildcard for prefix or suffix; surround term with wildcard characters for substring search. Ignored in phrase searching.
(	Begin logical grouping
)	End logical grouping
"	delimiter for phrase searching

The operators `AND`, `OR`, and `NOT` are case sensitive.

Search Kit performs Unicode normalization on query strings and on the text placed into indexes. It uses Unicode Normalization Form KC (NFKC, compatibility decomposition followed by canonical composition) as documented in Unicode Standard Annex #15. For example, the a-grave character, 'à', can be written as the

two Unicode characters (0x0061, 0x0300) or as the single Unicode character 0x00E0. Search Kit will normalize (0x0061, 0x0300) to 0x00E0. For more information on Unicode normalization, see http://unicode.org/reports/tr15 .

Search Kit further normalizes query strings and indexes by stripping diacritical marks and by forcing characters to lowercase. For example, Search Kit normalizes each of the following characters to 'a': 'a', 'à', 'A', and 'À'.

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

When your application no longer needs the search object, dispose of it by calling CFRelease.

Special Considerations

Search Kit supports logical exclusion. The NOT and ! operators behave as though they were EXCLUDE operators. For example, a search for 'red NOT blue' returns all documents that contain the word 'red' and do not contain the word 'blue'.

Unary Boolean operators, however, are not currently implemented in Search Kit. A search, for example, for 'NOT blue', returns zero documents no matter what their content.

You cannot use CFMakeCollectable with SKSearch objects. In a garbage-collected environment, you must use CFRelease to dispose of an SKSearch object.

Version Notes

Mac OS X version 10.4 uses a completely revised, and far more powerful, query approach than did earlier versions of Mac OS X. Refer to the Discussion in this function for details. Refer to SKSearchResultsCreateWithQuery (page 2772) (deprecated) for a description of Search Kit's behavior in earlier versions of Mac OS X.

In versions of Mac OS X prior to version 10.4, Search Kit did not perform Unicode normalization, and did not remove diacritical marks.

Availability

Available in Mac OS X v10.4 and later.

Declared In

SKSearch.h

SKSearchFindMatches

Extracts search result information from a search object.

```
Boolean SKSearchFindMatches (
    SKSearchRef         inSearch,
    CFIndex             inMaximumCount,
    SKDocumentID        *outDocumentIDsArray,
    float               *outScoresArray,
    CFTimeInterval      maximumTime
    CFIndex             *outFoundCount
);
```

Parameters

inSearch

A reference to a search object (SKSearchRef) previously created with SKSearchCreate.

inMaximumCount

The maximum number of items to find. For each item found, `SKSearchFindMatches` places the associated document ID into the *outDocumentIDsArray* array. Specify an *inMaximumCount* of 0 to find as many items as possible within *maximumTime*.

outDocumentIDsArray

On input, a pointer to an array for document IDs. On output, points to points to the previously allocated array, which now contains the found document IDs. The size of this array must be equal to *inMaximumCount*.

outScoresArray

On input, a pointer to an array for scores. On output, points to the previously allocated array, which now contains relevance scores for the found items. The size of this array, if not `NULL`, must be equal to *inMaximumCount*. Can be `NULL` on input, provided that your application doesn't need this information. Search Kit does not normalize relevance scores, so they can be very large.

maximumTime

The maximum number of seconds before this function returns, whether or not *inMaximumCount* items have been found. Setting maximumTime to 0 tells the search to return quickly

outFoundCount

On input, a pointer to a CFIndex object that will hold the number of items found. On output, points to the CFIndex object that now contains the actual number of items found.

Return Value
A logical value indicating whether the search is still in progress. Returns false when the search is exhausted.

Discussion
The `SKSearchFindMatches` extracts results from a find operation initiated by a search object (SKSearchRef).

This function provides results to its output parameters simply in the order in which they are found. This reduces latency to support search-as-you-type functionality. Larger scores mean greater relevance.

You can call this function on a search object repeatedly to get additional sets of search results. For example, if you call this function twice with an *inMaximumCount* value of 10, the first call will put the first 10 items found into the output arrays and the second call will put the second 10 items found into the output arrays.

Applications are free to display relevance scores in any appropriate manner. One simple way is to divide each relevance score by the largest number returned to get relevance numbers scaled linearly from 0.0 to 1.0. Search Kit does not scale the relevance scores for you, because you may want to combine the scores from several calls on a search object or the scores from calls to more than one search object.

Search Kit is thread-safe. You can use separate indexing and searching threads. Your application is responsible for ensuring that no more than one process is open at a time for writing to an index.

Before invoking a search, call `SKIndexFlush` (page 2750) on all indexes you will query to ensure that updates to the indexes have been flushed to disk.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`SKSearch.h`

SKSearchGetTypeID

Gets the type identifier for Search Kit search objects.

```
CFTypeID SKSearchGetTypeID (void);
```

Return Value
A CFTypeID object containing the type identifier for the SKSearch opaque type.

Discussion
Search Kit represents searches with search objects (`SKSearchRef` (page 2782) opaque types). If your code needs to determine whether a particular data type is a search object, you can use this function along with the `CFGetTypeID` function and perform a comparison.

Never hard-code the search type ID because it can change from one release of Mac OS X to another.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`SKSearch.h`

SKSearchGroupCopyIndexes

Obtains the indexes for a search group. (Deprecated in Mac OS X v10.4. Use asynchronous searching with SKSearchCreate instead, which does not employ search groups.)

```
CFArrayRef SKSearchGroupCopyIndexes (
    SKSearchGroupRef    inSearchGroup
);
```

Parameters

inSearchGroup
The search group whose indexes you want to copy.

Return Value
A CFArray object containing the indexes in the search group.

Discussion
Although the search functions `SKSearchResultsCreateWithQuery` (page 2772) and `SKSearchResultsCreateWithDocuments` (page 2770) operate directly on search groups, many Search Kit functions, such as `SKIndexCompact` (page 2741), operate on one index at a time. When you want to examine or manage all the indexes in a search group, use `SKSearchGroupCopyIndexes` to get the search group's list of indexes.

Availability
Available in Mac OS X v10.3 and later.
Deprecated in Mac OS X v10.4.

Declared In
`SKSearch.h`

SKSearchGroupCreate

Creates a search group as an array of references to indexes. (Deprecated in Mac OS X v10.4. Use asynchronous searching with SKSearchCreate instead, which does not employ search groups.)

```
SKSearchGroupRef SKSearchGroupCreate (
    CFArrayRef      inArrayOfInIndexes
);
```

Parameters

inArrayOfInIndexes

A CFArray object containing the indexes to put into the search group.

Return Value

An SKSearchGroup opaque type.

Discussion

Creates a search group as an array of references to indexes.

You create a search group to search one or more indexes, and then typically use the resulting `SKSearchGroupRef` opaque type with `SKSearchResultsCreateWithQuery` (page 2772) or `SKSearchResultsCreateWithDocuments` (page 2770).

When your application no longer needs the search group, dispose of it by calling `CFRelease`.

Availability

Available in Mac OS X v10.3 and later.

Deprecated in Mac OS X v10.4.

Declared In

`SKSearch.h`

SKSearchGroupGetTypeID

Deprecated. Use asynchronous searching with SKSearchCreate instead, which does not employ search groups.

```
CFTypeID SKSearchGroupGetTypeID (void);
```

Return Value

A CFTypeID object containing the type identifier for the SKSearchGroup opaque type.

Discussion

Gets the type identifier for Search Kit search groups.

Search Kit represents search groups with the `SKSearchGroupRef` (page 2784) opaque type. If your code needs to determine whether a particular data type is a search group, you can use this function along with the `CFGetTypeID` function and perform a comparison.

Never hard-code the search group type ID because it can change from one release of Mac OS X to another.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`SKSearch.h`

SKSearchResultsCopyMatchingTerms

Obtains the terms in a document that match a query. (Deprecated in Mac OS X v10.4. Use SKSearchCreate (page 2764) instead.)

```
CFArrayRef SKSearchResultsCopyMatchingTerms (
    SKSearchResultsRef  inSearchResults,
    CFIndex             inItem
);
```

Parameters

inSearchResults

The search results to examine.

inItem

An integer that corresponds to a document URL object (SKDocumentRef) in the search results. A value of '1' identifies the first document URL object in the search results, a value of '2' identifies the second, and so on.

If you've created the search results using SKSearchResultsCreateWithQuery (page 2772), the document URL objects are sorted in ranking order with the top-ranked one first. See SKSearchResultsGetInfoInRange (page 2773) for a description of how to get a particular document URL object, or set of them, from a search result.

Return Value

A CFArray object containing term IDs.

Discussion

When using a prefix search, or a search for which the user entered more than one word, there may be multiple terms that match the query. This function returns an array of the term IDs corresponding to these matches.

For example, a user could enter 'App' when performing a prefix search. If a document represented in the search group contains the words 'Apple,' 'application,' and 'appendectomy,' the IDs for all of these terms would then appear in the CFArray object that SKSearchResultsCopyMatchingTerms returns.

See SKSearchResultsCreateWithQuery (page 2772) for a description of how to perform a search and get search results. See SKSearchResultsGetInfoInRange (page 2773) for how to extract information, including document URL objects, from a search result. See "Deprecated Search Keys" (page 2789) for a description of the various categories of search.

Availability

Available in Mac OS X v10.3 and later.

Deprecated in Mac OS X v10.4.

Declared In

SKSearch.h

SKSearchResultsCreateWithDocuments

Finds documents similar to given example documents. (Deprecated in Mac OS X v10.4. Use SKSearchCreate (page 2764) instead.)

```
SKSearchResultsRef SKSearchResultsCreateWithDocuments (
    SKSearchGroupRef                inSearchGroup,
    CFArrayRef                      inExampleDocuments,
    CFIndex                         inMaxFoundDocuments,
    void                            *inContext,
    SKSearchResultsFilterCallBack   inFilterCallBack
);
```

Parameters

inSearchGroup

A search group containing the indexes which, in turn, contain the document URL objects (SKDocumentRefs) representing the documents you want to search by similarity. The search group must also contains the indexes that contain the textual content of the example documents.

inExampleDocuments

An array of document URL objects (SKDocumentRefs), each representing an example document.

inMaxFoundDocuments

The maximum number of found items to return. Your application must pass in a positive value.

inContext

An application-specified context for use by the `SKSearchResultsFilterCallBack` (page 2780) callback function. Can be `NULL`.

inFilterCallBack

A callback function for hit testing during searching—see `SKSearchResultsFilterCallBack` (page 2780). In a similarity search, your application would typically use this function to exclude the example documents from the search results. This parameter can be `NULL`, in which case your application receives the returned results directly and without any custom postprocessing.

Return Value

A search results object containing a list of document URL objects (SKDocumentRefs) representing documents similar to the example documents.

Discussion

This function searches the on-disk indexes in a search group for document URL objects (SKDocumentRefs) representing documents similar to those provided as examples. Build the search group in three steps:

1. Collect the index IDs from the search groups you want to search: for each search group, call the `SKSearchGroupCopyIndexes` (page 2768) function.

2. Add the document URL objects representing the example documents to a memory-based index (if they're not already in an index) by calling `SKIndexCreateWithMutableData` (page 2746), and get that index's ID.

3. Create a new search group that contains the indexes to search, and also containing the example-documents index, using `SKSearchGroupCreate` (page 2769).

Before invoking a search, call `SKIndexFlush` (page 2750) on all indexes in the search group to ensure that changes to the indexes have been written to disk.

Once you've obtained the results of a search, get the specifics—including which documents match the user's similarity query, and the ranking scores for each document—by calling `SKSearchResultsGetInfoInRange` (page 2773).

When your application no longer needs the search result, dispose of it by calling `CFRelease`.

Availability
Available in Mac OS X v10.3 and later.
Deprecated in Mac OS X v10.4.

Declared In
SKSearch.h

SKSearchResultsCreateWithQuery

Queries the indexes in a search group. (Deprecated in Mac OS X v10.4. Use SKSearchCreate (page 2764) instead.)

```
SKSearchResultsRef SKSearchResultsCreateWithQuery (
    SKSearchGroupRef        inSearchGroup,
    CFStringRef             inQuery,
    SKSearchType            inSearchType,
    CFIndex                 inMaxFoundDocuments,
    void                    *inContext,
    SKSearchResultsFilterCallBackinFilterCallBack
);
```

Parameters

inSearchGroup
The search group to query.

inQuery
The query string to search for.

inSearchType
The category of search to perform. See the "Deprecated Search Keys" (page 2789) enumeration for options.

inMaxFoundDocuments
The maximum number of found items to return. Your application must pass in a positive integer value.

inContext
An application-specified context for use by the SKSearchResultsFilterCallBack (page 2780). Can be NULL, but if you want to use the callback you must supply a context.

inFilterCallBack
A callback function for hit testing during searching. Can be NULL, in which case your application receives the returned results directly and without any custom postprocessing. If non-NULL, you must supply a context. See SKSearchResultsFilterCallBack (page 2780).

Return Value
A search results object.

Discussion
This function searches the on-disk indexes in a search group. Before invoking a search, call SKIndexFlush (page 2750) on all indexes in the search group to ensure that changes to the indexes have been flushed to disk.

Once you've obtained the results of a search, get the specifics—including which documents match the user's query, and the ranking scores for each document—by calling SKSearchResultsGetInfoInRange (page 2773). You can extract other information by calling SKSearchResultsCopyMatchingTerms (page 2770) and SKSearchResultsGetCount (page 2773).

When your application no longer needs the search result, dispose of it by calling `CFRelease`.

Special Considerations

This deprecated function performs searches synchronously. Apple recommends using the asynchronous `SKSearchCreate` function instead.

In the current implementation of Search Kit, unary Boolean operators are not implemented. A search, for example, for 'not blue', returns zero documents no matter what their content.

Availability
Available in Mac OS X v10.3 and later.
Deprecated in Mac OS X v10.4.

Declared In
`SKSearch.h`

SKSearchResultsGetCount

Gets the total number of found items in a search. (Deprecated in Mac OS X v10.4. Use `SKSearchCreate` (page 2764) instead.)

```
CFIndex SKSearchResultsGetCount (
    SKSearchResultsRef  inSearchResults
);
```

Parameters

`inSearchResults`
A search results object containing the results of a query.

Return Value
A CFIndex object containing the total number of found items in a search.

Availability
Available in Mac OS X v10.3 and later.
Deprecated in Mac OS X v10.4.

Declared In
`SKSearch.h`

SKSearchResultsGetInfoInRange

Extracts information from a Search Kit query result. (Deprecated in Mac OS X v10.4. Use `SKSearchCreate` (page 2764) instead.)

```
CFIndex SKSearchResultsGetInfoInRange (
    SKSearchResultsRef  inSearchResults,
    CFRange             inRange,
    SKDocumentRef       *outDocumentsArray,
    SKIndexRef          *outIndexesArray,
    float               *outScoresArray
);
```

Parameters

inSearchResults

The search results whose information you want to extract.

inRange

The starting ranking and total number of found items to obtain, specified as `(Location, Length)`. 'Location' specifies the starting item by ranking, with the top-ranked item having a location of 0. 'Length' specifies the total number of items to include in the results. For example, (0,1) indicates the first item, which is also the highest-ranking item. (1,1) indicates the second item, which is also the second-highest-ranking item. (0,5) means to get the first 5 items.

outDocumentsArray

On output, points to an array of found document URL objects (SKDocumentRefs).

outIndexesArray

On output, points to an array of indexes in which the found document URL objects reside. Can be `NULL` on input, provided that your application doesn't need this information.

outScoresArray

On output, points to an array of correspondence scores for found items. Can be `NULL` on input, provided that your application doesn't need this information.

Return Value

The number of items returned—usually the same number as specified by the length item in the `inRange` parameter.

Discussion

This function provides results to its output parameters in the order in which they are found, to reduce latency and to support search-as-you-type functionality.

Availability

Available in Mac OS X v10.3 and later.

Deprecated in Mac OS X v10.4.

Declared In

`SKSearch.h`

SKSearchResultsGetTypeID

Gets the type identifier for Search Kit search results. (**Deprecated.** Use `SKSearchCreate` (page 2764) instead.)

```
CFTypeID SKSearchResultsGetTypeID (void);
```

Return Value

A CFTypeID object containing the type identifier for the SKSearchResults opaque type.

Discussion
Search Kit represents search results with search results objects (`SKSearchResultsRef` (page 2783) opaque types). If your code needs to determine whether a particular data type is a search result, you can use this function along with the `CFGetTypeID` function and perform a comparison.

Never hard-code the search result type ID because it can change from one release of Mac OS X to another.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKSearch.h`

SKSummaryCopyParagraphAtIndex

Gets a specified paragraph from the text in a summarization object.

```
CFStringRef SKSummaryCopyParagraphAtIndex (
    SKSummaryRef        summary,
    CFIndex             i,
);
```

Parameters

summary
The summarization object containing the text from which you want a paragraph.

i
The ordinal number of the paragraph in the original text, with the first paragraph designated by zero (this function uses zero-based indexing).

Return Value
A CFString object containing the specified paragraph, or `NULL` on failure.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`SKSummary.h`

SKSummaryCopyParagraphSummaryString

Gets a text string consisting of a summary with, at most, the requested number of paragraphs.

```
CFStringRef SKSummaryCopyParagraphSummaryString (
    SKSummaryRef        summary,
    CFIndex             numParagraphs
);
```

Parameters

summary
The summarization object containing the text from which you want a summarization.

numParagraphs
The maximum number of paragraphs you want in the summary.

Return Value
A CFString object containing the requested summary.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`SKSummary.h`

SKSummaryCopySentenceAtIndex

Gets a specified sentence from the text in a summarization object.

```
CFStringRef SKSummaryCopySentenceAtIndex (
    SKSummaryRef        summary,
    CFIndex             i,
);
```

Parameters

summary
The summarization object containing the text from which you want a sentence.

i
The ordinal number of the sentence in the original text, with the first sentence designated by zero (this function uses zero-based indexing).

Return Value
A CFString object containing the specified sentence, or `NULL` on failure.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`SKSummary.h`

SKSummaryCopySentenceSummaryString

Gets a text string consisting of a summary with, at most, the requested number of sentences.

```
CFStringRef SKSummaryCopySentenceSummaryString (
    SKSummaryRef        summary,
    CFIndex             numSentences
);
```

Parameters

summary
The summarization object containing the text from which you want a summarization.

numSentences
The maximum number of sentences you want in the summary.

Return Value
A CFString object containing the requested summary.

Availability
Available in Mac OS X v10.4 and later.

Declared In
SKSummary.h

SKSummaryCreateWithString

Creates a summary object based on a text string.

```
SKSummaryRef SKSummaryCreateWithString (
    CFStringRef     inString
);
```

Parameters

inString
The text string that you want to summarize.

Return Value
Returns a summarization object, or NULL on failure.

Discussion
The SKSummaryCreateWithString function creates a summarization object that pre-analyzes a text string to support fast summarization. When your application no longer needs the summarization object, dispose of it by calling CFRelease.

Availability
Available in Mac OS X v10.4 and later.

Declared In
SKSummary.h

SKSummaryGetParagraphCount

Gets the number of paragraphs in a summarization object.

```
CFIndex SKSummaryGetParagraphCount (
    SKSummaryRef    summary
);
```

Parameters

summary
The summarization object whose paragraphs you want to count.

Return Value
A CFIndex object containing the number of paragraphs in the summarization object.

Availability
Available in Mac OS X v10.4 and later.

Declared In
SKSummary.h

SKSummaryGetParagraphSummaryInfo

Gets detailed information about a body of text for constructing a custom paragraph-based summary string.

```
CFIndex SKSummaryGetParagraphSummaryInfo (
    SKSummaryRef            summary,
    CFIndex                 numParagraphsInSummary,
    CFIndex                 *outRankOrderOfParagraphs,
    CFIndex                 *outParagraphIndexOfParagraphs
);
```

Parameters

summary

The summarization object containing the text from which you want to build a summary.

numParagraphsInSummary

The maximum number of paragraphs you want in the summary.

outRankOrderOfParagraphs

On input, a pointer to an array of CFIndex objects. On output, points to the previously allocated array, which now lists the summarization relevance rank of each paragraph in the original text. The most important paragraph gets a rank of 1. The array size must equal *numParagraphsInSummary*, or else be NULL if you don't want to get the relevance ranks.

outParagraphIndexOfParagraphs

On output, points to an array containing the ordinal number for each paragraph in the original text. Use the SKSummaryCopyParagraphAtIndex (page 2775) function with one of these numbers to get the corresponding paragraph. The array size must equal *numParagraphsInSummary*, or else be NULL if you don't want to get the ordinal numbers of the paragraphs.

Return Value
The number of paragraphs in the summary.

Availability
Available in Mac OS X v10.4 and later.

Declared In
SKSummary.h

SKSummaryGetSentenceCount

Gets the number of sentences in a summarization object.

```
CFIndex SKSummaryGetSentenceCount (
    SKSummaryRef    summary
);
```

Parameters

summary

The summarization object whose sentences you want to count.

Return Value
A CFIndex object containing the number of sentences in the summarization object.

Availability
Available in Mac OS X v10.4 and later.

Declared In
SKSummary.h

SKSummaryGetSentenceSummaryInfo

Gets detailed information about a body of text for constructing a custom sentence-based summary string.

```
CFIndex SKSummaryGetSentenceSummaryInfo (
    SKSummaryRef            summary,
    CFIndex                 numSentencesInSummary,
    CFIndex                 *outRankOrderOfSentences,
    CFIndex                 *outSentenceIndexOfSentences,
    CFIndex                 *outParagraphIndexOfSentences
);
```

Parameters

summary

The summarization object containing the text from which you want to build a summary.

numSentencesInSummary

The maximum number of sentences you want in the summary.

outRankOrderOfSentences

On input, a pointer to an array of CFIndex objects. On output, points to the previously allocated array, which now lists the summarization relevance rank of each sentence in the original text. The most important sentence gets a rank of 1. The array size must equal *numSentencesInSummary*, or else be NULL if you don't want to get the rank orders.

outSentenceIndexOfSentences

On input, a pointer to an array of CFIndex objects. On output, points to the previously allocated array, which now contains the ordinal number for each sentence in the original text. Use the SKSummaryCopySentenceAtIndex (page 2776) function with one of these numbers to get the corresponding sentence. The array size must equal *numSentencesInSummary*, or else be NULL if you don't want to get the ordinal numbers of the sentences.

outParagraphIndexOfSentences

On input, a pointer to an array of CFIndex objects. On output, points to the previously allocated array, which now contains the ordinal number for the paragraph that each corresponding sentence, referenced in *outSentenceIndexOfSentences*, appears in. The array size must equal *numSentencesInSummary*, or else be NULL if you don't want to get the ordinal numbers of the sentences.

Return Value

The number of sentences in the summary.

Availability

Available in Mac OS X v10.4 and later.

Declared In

SKSummary.h

SKSummaryGetTypeID

Gets the type identifier for Search Kit summarization objects.

```
CFTypeID SKSummaryGetTypeID (void);
```

Return Value

A CFTypeID object, or NULL on failure.

Discussion
Search Kit represents summarization results with summarization objects (`SKSummaryRef` (page 2783) opaque types). If your code needs to determine whether a particular data type is a summary, you can use this function along with the `CFGetTypeID` function and perform a comparison.

Never hard-code the summarization type ID because it can change from one release of Mac OS X to another.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`SKSummary.h`

Callbacks

Developers should avoid using the callbacks listed in this section; instead, use `SKSearchCreate` (page 2764) and `SKSearchFindMatches` (page 2766).

SKSearchResultsFilterCallBack

Deprecated. Use `SKSearchCreate` and `SKSearchFindMatches` instead, which do not use a callback.

```
typedef Boolean (SKSearchResultsFilterCallBack) (
    SKIndexRef     inIndex,
    SKDocumentRef     inDocument,
    void      *inContext
```

If you name your function `MySearchResultsFilter`, you would declare it like this:

```
Boolean MySearchResultsFilter (
    SKIndexRef     inIndex,
    SKDocumentRef     inDocument,
    void      *inContext
);
```

Parameters

inIndex
The index you are searching.

inDocument
The document URL object within the index you are searching.

inContext
An application-specified context which you set when calling `SKSearchResultsCreateWithQuery` (page 2772) or `SKSearchResultsCreateWithDocuments` (page 2770).

Return Value
A Boolean value of `true` for a successful search hit, or `false` otherwise.

Discussion
Deprecated. Defines a pointer to a search-results filtering callback function for hit testing and processing during a search. Use this callback function to perform custom filtering on the search hits returned by the `SKSearchResultsCreateWithQuery` (page 2772) and `SKSearchResultsCreateWithDocuments` (page 2770) functions. Return `true` to keep this document URL object (SKDocumentRef) in the results, `false` to filter it out.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKSearch.h`

Data Types

SKDocumentRef

Defines an opaque data type representing a document's URL.

```
typedef struct __SKDocument *SKDocumentRef;
```

Discussion
A document URL object is a generic location specification for a document. It is built from a document scheme, a parent document, and a document name. You can convert back and forth between document URL objects and `CFURL` objects using Search Kit's `SKDocumentCreateWithURL` (page 2735) and `SKDocumentCopyURL` (page 2734) functions.

To create a Search Kit document URL object, use `SKDocumentCreateWithURL` (page 2735) when you can provide a complete URL, or use `SKDocumentCreate` (page 2734) when you want to specify document location indirectly using a parent document URL object. For other operations on documents, see "Working with Documents and Terms" (page 2731).

If you create document URL objects with indirect locations using the `SKDocumentCreate` (page 2734) function, you can resolve the locations by assembling them piece by piece, starting with a document URL object and going up step by step, parent to parent.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKDocument.h`

SKIndexDocumentIteratorRef

Defines an opaque data type representing an index-based document iterator.

```
typedef struct __SKIndexDocumentIterator *SKIndexDocumentIteratorRef;
```

Discussion
A Search Kit document iterator lets your application loop through all the document URL objects owned by a given parent document URL object. To create an iterator, use `SKIndexDocumentIteratorCreate` (page 2749). To get a copy of the next document in the set owned by the iterator, use `SKIndexDocumentIteratorCopyNext` (page 2748).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKIndexRef

Defines an opaque data type representing an index.

```
typedef struct __SKIndex *SKIndexRef;
```

Discussion
A Search Kit index object contains the textual contents of one or more documents, as well as document URL objects (SKDocumentRefs) representing those documents' locations.

To create a new disk-based Search Kit index object, use `SKIndexCreateWithURL` (page 2747). To create a memory-based index, use `SKIndexCreateWithMutableData` (page 2746). For other operations on indexes, see "Creating, Opening, and Closing Indexes" (page 2729) and "Managing Indexes" (page 2730). Also see"Fast Asynchronous Searching" (page 2732).

Special Considerations

You cannot use `CFMakeCollectable` with SKIndex objects. In a garbage-collected environment, you must use `SKIndexClose` (page 2740) to dispose of an SKIndex object.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKIndex.h`

SKSearchRef

Defines an opaque data type representing a an asynchronous search.

```
typedef struct __SKSearch *SKSearchRef;
```

Discussion
A search object is created when you call the `SKSearchCreate` (page 2764) function.

Special Considerations

You cannot use `CFMakeCollectable` with SKSearch objects. In a garbage-collected environment, you must use `CFRelease` to dispose of an SKSearch object.

Availability
Available in Mac OS X v10.4 and later.

Declared In
SKSearch.h

SKSummaryRef

Defines an opaque data type representing summarization information.

```
typedef struct __SKSummary *SKSummaryRef;
```

Discussion
A summarization object contains summarization information, including summary text.

Availability
Available in Mac OS X v10.4 and later.

Declared In
SKSummary.h

SKDocumentID

Defines an opaque data type representing a lightweight document identifier.

```
typedef CFIndex SKDocumentID;
```

Discussion
Use document IDs rather than document URL objects (SKDocumentRefs) whenever possible. Using document IDs results in faster searching.

You can work with document IDs using a variety of Search Kit functions. See SKIndexGetMaximumDocumentID (page 2755), SKIndexCopyDocumentForDocumentID (page 2741), SKIndexCopyInfoForDocumentIDs (page 2744), SKIndexCopyDocumentRefsForDocumentIDs (page 2743), SKIndexCopyDocumentURLsForDocumentIDs (page 2744), SKIndexCopyDocumentIDArrayForTermID (page 2742), and SKIndexCopyTermIDArrayForDocumentID (page 2745).

Availability
Available in Mac OS X v10.4 and later.

Declared In
SKIndex.h

SKSearchResultsRef

Deprecated. Use asynchronous searching with SKSearchCreate instead, which does not employ search groups.

```
typedef struct __SKSearchResults *SKSearchResultsRef;
```

Discussion
Defines an opaque data type representing the result of a search. To perform a query and generate search results, use `SKSearchResultsCreateWithQuery` (page 2772) or `SKSearchResultsCreateWithDocuments` (page 2770). To examine the result of a search, use `SKSearchResultsGetInfoInRange` (page 2773). For other operations on search results, see "Legacy Support for Synchronous Searching" (page 2733).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKSearch.h`

SKSearchGroupRef

Deprecated. Use asynchronous searching with SKSearchCreate instead, which does not employ search groups.

```
typedef struct __SKSearchGroup *SKSearchGroupRef;
```

Discussion
Defines an opaque data type representing a search group.

A search group is a group of one or more indexes to be searched. To create a search group, use `SKSearchGroupCreate` (page 2769). For other operations with search groups, see "Fast Asynchronous Searching" (page 2732).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SKSearch.h`

Constants

Text Analysis Keys

Each of these constants is an optional key in a Search Kit index's text analysis properties dictionary. The constant descriptions describe the corresponding values for each of these keys. These keys are declared in the `Analysis.h` header file.

```
const CFStringRef kSKMinTermLength;
const CFStringRef kSKStopWords;
const CFStringRef kSKSubstitutions;
const CFStringRef kSKMaximumTerms;
const CFStringRef kSKProximityIndexing;
const CFStringRef kSKTermChars;
const CFStringRef kSKStartTermChars;
const CFStringRef kSKEndTermChars;
```

Constants

`kSKMinTermLength`

The minimum term length to index. Specified as a CFNumber object. If this optional key is not present, Search Kit indexing defaults to a minimum term length of 1.

Available in Mac OS X v10.3 and later.

Declared in `SKAnalysis.h`.

`kSKStopWords`

A set of stopwords—words not to index. Specified as a CFSet object. There is no default stopword list. You must supply your own.

Available in Mac OS X v10.3 and later.

Declared in `SKAnalysis.h`.

`kSKSubstitutions`

A dictionary of term substitutions—terms that differ in their character strings but that match during a search. Specified as a CFDictionary object.

Available in Mac OS X v10.3 and later.

Declared in `SKAnalysis.h`.

`kSKMaximumTerms`

The maximum number of number unique terms to index in each document. Specified as a CFNumber object.

Search Kit indexes from the beginning of a document. When it has indexed the first n unique terms, it stops.

The default number of maximum terms, which applies if you do not provide a number, is 2000.

To tell Search Kit to index all the terms in each document without limit, specify a value of 0.

Available in Mac OS X v10.4 and later.

Declared in `SKAnalysis.h`.

`kSKProximityIndexing`

A Boolean flag indicating whether or not Search Kit should use proximity indexing. The flag can be a `0` or `kCFBoolenFalse` value (for false) or a `1` or `kCFBooleanTrue` value for true.

Proximity indexing supports phrase searching. If this key is not present in an index's text analysis properties dictionary, Search Kit defaults to not adding proximity information to the index.

Available in Mac OS X v10.4 and later.

Declared in `SKAnalysis.h`.

`kSKTermChars`

Additional valid starting-position "word" characters for indexing and querying. The corresponding value, a CFString object, specifies the additional valid "word" characters that you want to be considered as valid starting characters of terms for indexing and querying. "Word" characters are contrasted with nonword characters, such as spaces.

The value of `kSKStartTermChars`, if this key is present, overrides the value of `kSKTermChars` for the first character of a term.

By default, Search Kit considers alphanumeric characters as valid starting characters for terms, and considers all others (including the underscore character) to be nonword characters.

Available in Mac OS X v10.4 and later.

Declared in `SKAnalysis.h`.

`kSKStartTermChars`

Additional valid starting-position "word" characters for indexing and querying. The corresponding value, a CFString object, specifies the additional valid "word" characters that you want to be considered as valid starting characters of terms for indexing and querying. "Word" characters are contrasted with nonword characters, such as spaces.

The value of `kSKStartTermChars`, if this key is present, overrides the value of `kSKTermChars` for the first character of a term.

By default, Search Kit considers alphanumeric characters as valid starting characters for terms, and considers all others (including the underscore character) to be nonword characters.

Available in Mac OS X v10.4 and later.

Declared in `SKAnalysis.h`.

`kSKEndTermChars`

Additional valid last-position "word" characters for indexing and querying. The corresponding value, a CFString object, specifies the additional valid "word" characters that you want to be considered as valid ending characters of terms for indexing and querying. "Word" characters are contrasted with nonword characters, such as spaces.

The value of `kSKEndTermChars`, if this key is present, overrides the value of `kSKTermChars` for the last character of a term.

By default, Search Kit considers alphanumeric characters as valid ending characters for terms, and considers all others (including the underscore character) to be nonword characters.

Available in Mac OS X v10.4 and later.

Declared in `SKAnalysis.h`.

SKDocumentIndexState

The indexing state of a document.

```
enum SKDocumentIndexState {
    kSKDocumentStateNotIndexed  = 0,
    kSKDocumentStateIndexed     = 1,
    kSKDocumentStateAddPending  = 2,
    kSKDocumentStateDeletePending= 3
};
```

Constants

`kSKDocumentStateNotIndexed`

Specifies that the document is not indexed.

Available in Mac OS X v10.3 and later.

Declared in `SKIndex.h`.

`kSKDocumentStateIndexed`

Specifies that the document is indexed.

Available in Mac OS X v10.3 and later.

Declared in `SKIndex.h`.

`kSKDocumentStateAddPending`

Specifies that the document is not in the index but will be added after the index is flushed or closed.

Available in Mac OS X v10.3 and later.

Declared in `SKIndex.h`.

`kSKDocumentStateDeletePending`

Specifies that the document is in the index but will be deleted after the index is flushed or closed.

Available in Mac OS X v10.3 and later.

Declared in `SKIndex.h`.

Declared In

`SKIndex.h`

SKSearchOptions

Specifies the search options available for the `SKSearchCreate` (page 2764) function.

```
typedef UInt32 SKSearchOptions;
enum SKSearchType {
    kSKSearchOptionDefault = 0,
    kSKSearchOptionNoRelevanceScores = 1L << 0,
    kSKSearchOptionSpaceMeansOR = 1L << 1
    kSKSearchOptionFindSimilar = 1L << 2
};
```

Constants

`kSKSearchOptionDefault`

Default search options include:

- Relevance scores will be computed
- Spaces in a query are interpreted as Boolean `AND` operators.
- Do not use similarity searching.

Available in Mac OS X v10.4 and later.

Declared in `SKSearch.h`.

`kSKSearchOptionNoRelevanceScores`
: This option saves time during a search by suppressing the computation of relevance scores.

  Available in Mac OS X v10.4 and later.

  Declared in `SKSearch.h`.

`kSKSearchOptionSpaceMeansOR`
: This option alters query behavior so that spaces are interpreted as Boolean `OR` operators.

  Available in Mac OS X v10.4 and later.

  Declared in `SKSearch.h`.

`kSKSearchOptionFindSimilar`
: This option alters query behavior so that Search Kit returns references to documents that are similar to an example text string. When this option is specified, Search Kit ignores all query operators.

  Available in Mac OS X v10.4 and later.

  Declared in `SKSearch.h`.

Declared In
`SKSearch.h`

SKIndexType

Specifies the category of an index.

```
enum SKIndexType {
    kSKIndexUnknown         = 0,
    kSKIndexInverted        = 1,
    kSKIndexVector          = 2,
    kSKIndexInvertedVector  = 3
};
```

Constants
`kSKIndexUnknown`
: Specifies an unknown index type.

  Available in Mac OS X v10.3 and later.

  Declared in `SKIndex.h`.

`kSKIndexInverted`
: Specifies an inverted index, mapping terms to documents.

  Available in Mac OS X v10.3 and later.

  Declared in `SKIndex.h`.

`kSKIndexVector`
: Specifies a vector index, mapping documents to terms.

  Available in Mac OS X v10.3 and later.

  Declared in `SKIndex.h`.

`kSKIndexInvertedVector`
: Specifies an index type with all the capabilities of an inverted and a vector index.

  Available in Mac OS X v10.3 and later.

  Declared in `SKIndex.h`.

Declared In
`SKIndex.h`

Deprecated Text Analysis Keys

Search Kit ignores the `kSKLanguageTypes` constant. It determines language directly by document content.

```
const CFStringRef kSKLanguageTypes;
```

Constants

`kSKLanguageTypes`

Deprecated—Search Kit ignores this constant.

In releases of Mac OS X previous to version 10.4, each string in this key's corresponding value specifies a language to use for indexing. Each such string is a two character ISO 639-1 code. For example, `'en'` for English, `'ja'` for Japanese, and so on. If this key is not present, Search Kit uses the Mac OS X preferences system to determine the primary language from the user's locale.

Available in Mac OS X v10.3 and later.

Declared in `SKAnalysis.h`.

Version Notes

In releases of Mac OS X prior to version 10.4, the `kSKLanguageTypes` constant was an optional key in an index's text analysis properties dictionary. Starting in Mac OS X v10.4, Search Kit ignores this constant and determines language directly by the document content. A document may use multiple languages.

Deprecated Search Keys

Search Kit ignores the constants in this group. Use asynchronous searching with `SKSearchCreate` instead, which uses query syntax to determine search type.

```
enum SKSearchType {
    kSKSearchRanked         = 0,
    kSKSearchBooleanRanked  = 1,
    kSKSearchRequiredRanked = 2,
    kSKSearchPrefixRanked   = 3
};
```

Constants

`kSKSearchRanked`

Deprecated. Specifies a basic ranked search.

Available in Mac OS X v10.3 and later.

Declared in `SKSearch.h`.

`kSKSearchBooleanRanked`

Deprecated. Specifies a query that can include Boolean operators including `'|'`, `'&'`, `'!'`, `'('`, and `')'`.

Available in Mac OS X v10.3 and later.

Declared in `SKSearch.h`.

`kSKSearchRequiredRanked`

Deprecated. Specifies a query that can include required (`'+'`) or excluded (`'-'`) terms.

Available in Mac OS X v10.3 and later.

Declared in `SKSearch.h`.

`kSKSearchPrefixRanked`

Deprecated. Specifies a prefix-based search, which matches terms that begin with the query string.

Available in Mac OS X v10.3 and later.

Declared in `SKSearch.h`.

Version Notes

In releases of Mac OS X prior to version 10.4, these constants specify the category of search to perform. Starting with Mac OS X v10.4, use asynchronous searching with `SKSearchCreate` instead, which uses query syntax to determine search type.

In older versions of Mac OS X, these constants specify the various search types you can use with `SKSearchResultsCreateWithQuery`. Each of these specifies a set of ranked search hits. The `kSKSearchRanked` and `kSKSearchPrefixRanked` constants can be used for all index types. The `kSKSearchBooleanRanked` and `kSKSearchRequiredRanked` constants cannot be used for vector indexes.

Declared In

`SKSearch.h`

Spotlight Metadata Attributes

Common Metadata Attribute Keys

Metadata attribute keys that are common to many file types.

kMDItemAttributeChangeDate

Date and time of the last change made to a metadata attribute.

Value Type: CFDate

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemAudiences

The audience for which the file is intended. The audience may be determined by the creator or the publisher or by a third party.

Value Type: Array of CFStrings

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemAuthors

The author, or authors, of the contents of the file. The order of the authors is preserved, but does not represent the main author or relative importance of the authors.

Value Type: Array of CFStrings

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemCity

Identifies city of origin according to guidelines established by the provider. For example, "New York", "Cupertino", or "Toronto".

Value Type: CFString

Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemComment

A comment related to the file. This comment is not displayed by the Finder.

Value Type: CFString
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemContactKeywords

A list of contacts that are associated with this document, not including the authors.

Value Type: Array of CFStrings
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemContentCreationDate

The date and time that the content was created.

Value Type: CFDate
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemContentModificationDate

Date and time when the content of this item was modified.

Value Type: CFDate
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemContentType

Uniform Type Identifier of the file. For example, a jpeg image file will have a value of public.jpeg. The value of this attribute is set by the Spotlight importer. Changes to this value are lost when the file attributes are next imported.

This attribute is marked as nosearch. You must specify this attribute key explicitly in a query in order for its contents to be searched.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemContentTypeTree

Uniform Type Identifier hierarchy of the file. For example, a jpeg image file will return an array containing "public.jpeg", "public.image", and "public.data". The value of this attribute is set by the Spotlight importer. Changes to this value are lost when the file attributes are next imported.

This attribute is marked as nosearch. You must specify this attribute key explicitly in a query in order for its contents to be searched.

Value Type: Array of CFStrings

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemContributors

Entities responsible for making contributions to the content of the resource. Examples of a contributor include a person, an organization or a service.

Value Type: Array of CFStrings

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemCopyright

Copyright owner of the file contents.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemCountry

The full, publishable name of the country or primary location where the intellectual property of the item was created, according to guidelines of the provider.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemCoverage

Extent or scope of the content of the resource. Coverage will typically include spatial location (a place name or geographic co-ordinates), temporal period (a period label, date, or date range) or jurisdiction (such as a named administrative entity). Recommended best practice is to select a value from a controlled vocabulary, and that, where appropriate, named places or time periods be used in preference to numeric identifiers such as sets of co-ordinates or date ranges.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemCreator

Name of the application used to create the document content. For example, "Pages" or "Keynote".

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemDescription

Description of the kind of item this file represents.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemDisplayName

Localized version of the file name. This is the localized version of the LaunchServices call `LSCopyDisplayNameForURL()`/ `LSCopyDisplayNameForRef()`.

Value Type: CFString
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemDueDate

Date this item is due.

Value Type: CFDate
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemDurationSeconds

The duration, in seconds, of the content of the item. A value of 10.5 represents media that is 10 and 1/2 seconds long.

Value Type: CFNumber
Units: seconds
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemEmailAddresses

Email addresses related to this item.

Value Type: Array of CFStrings
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemEncodingApplications

Applications used to convert the original content into it's current form. For example, a PDF file might have an encoding application set to "Distiller".

Value Type: Array of CFStrings
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemFinderComment

Finder comments for this item.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemFonts

Fonts used by this item. You should store the font's full name, the postscript name, or the font family name, based on the available information.

Value Type: Array of CFStrings

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemHeadline

Publishable entry providing a synopsis of the contents of the item. For example, "Apple Introduces the iPod Photo".

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemIdentifier

Formal identifier used to reference the resource within a given context. For example, the Message-ID of a mail message.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemInstantMessageAddresses

Instant message addresses related to this item.

Value Type: Array of CFStrings

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemInstructions

Instructions concerning the use of the item, such as embargoes and warnings. For example, "Second of four stories".

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemKeywords

Keywords associated with this file. For example, "Birthday", "Important", etc.

Value Type: Array of CFStrings

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemKind

Description of the kind of item this file represents.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemLanguages

Indicates the languages used by the item. The recommended best practice for the values of this attribute are defined by RFC 3066.

Value Type: Array of CFStrings

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemLastUsedDate

Date and time that the file was last used. This value is updated automatically by LaunchServices everytime a file is opened by double clicking, or by asking LaunchServices to open a file.

Value Type: CFDate
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemNumberOfPages

Number of pages in the document.

Value Type: CFNumber
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemOrganizations

Companies or organizations that created the document.

Value Type: Array of CFStrings
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemPageHeight

Height of the document page, in points (72 points per inch). For PDF files this indicates the height of the first page only.

Value Type: CFNumber
Units: points
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemPageWidth

Width of the document page, in points (72 points per inch). For PDF files this indicates the width of the first page only.

Value Type: CFNumber

Units: points

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemPhoneNumbers

Phone numbers related to this item.

Value Type: Array of CFStrings

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemProjects

List of projects related to this item. For example, if you were working on a movie, all of the files could be marked as belonging to the project "My Movie".

Value Type: Array of CFStrings

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemPublishers

Publishers of the item. For example, a person, an organization, or a service.

Value Type: Array of CFStrings

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemRecipients

Recipients of this item.

Value Type: Array of CFStrings

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemRights

Provides a link to information about rights held on the document. Contains a rights management statement for the document, or reference a service providing such information. Rights information often encompasses Intellectual Property Rights (IPR), copyright, and various property rights. If this attribute is absent, no assumptions can be made about the status of these and other rights with respect to the document.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemSecurityMethod

Encryption method used to make the item secure. PDF files return "None" or "Password Encrypted".

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemStarRating

User rating of this item. For example, the user rating (number of stars) of an iTunes track.

Value Type: CFNumber

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemStateOrProvince

Province or state of origin according to guidelines established by the provider. For example, "CA", "Ontario", or "Sussex".

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemTextContent

Contains a text representation of the content of the document. Data in multiple fields should be combined using a whitespace character as a separator. An application's Spotlight importer provides the content of this attribute.

Applications can create queries using this attribute, but are not able to read the value of this attribute directly.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemTitle

Title of the item. For example, this could be the title of a document, the name of an song, or the subject of an email message.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemVersion

Version number of the item.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemWhereFroms

Describes where the item was obtained from. For example, a downloaded file may refer to the URL, files received by email may indicate the sender's email address, message subject, etc.

Value Type: Array of CFStrings

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

Image Metadata Attribute Keys

Metadata attribute keys that are common to image files.

kMDItemAcquisitionMake

Manufacturer of the device used to acquire the document contents.

Value Type: CFString
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemAcquisitionModel

Model of the device used to acquire the document contents.

Value Type: CFString
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemAlbum

Title for the collection containing this item. This is analogous to a record label or photo album.

Value Type: CFString
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemAperture

Aperture setting used to aqure the document contents. This unit is the APEX value.

Value Type: CFNumber
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemBitsPerSample

Number of bits per sample. For example, the bit depth of an image (8-bit, 16-bit etc...) or the bit depth per audio sample of uncompressed audio data (8, 16, 24, 32, 64, etc..).

Value Type: CFNumber
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemColorSpace

Color space model used by the document contents. For example, “RGB”, “CMYK”, “YUV”, or “YCbCr”.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemEXIFVersion

Version of the EXIF header used to generate the metadata.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemExposureMode

Exposure mode used to acquire the document contents.

Value Type: CFNumber

Expected Values: 0 (auto exposure), 1 (manual exposure), 2 (auto bracket)

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemExposureProgram

Type of exposure program used by the camera to acquire the document contents. Possible values include: Manual, Normal, Aperture priority, etc.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemExposureTimeSeconds

Exposure time used to capture the document contents.

Value Type: CFNumber

Units: seconds

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemExposureTimeString

Time when the document contents were captured. Typically this corresponds to when a photograph is exposed.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemFNumber

Diameter of the aperture relative to the effective focal length of the lens.

Value Type: CFNumber

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemFlashOnOff

Whether a camera flash was used to capture the document contents.

Value Type: CFBoolean

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemFocalLength

Actual focal length of the lens, in millimeters.

Value Type: CFNumber

Units: millimeters

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemHasAlphaChannel

Whether the image has an alpha channel.

Value Type: CFBoolean
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemISOSpeed

ISO speed used to acquire the document contents. For example, 100, 200, 400, etc.

Value Type: CFNumber
Units: ISO Speed
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemLayerNames

Names of the layers in the file.

Value Type: Array of CFStrings
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemMaxAperture

Smallest F number of the lens in APEX value units, usually in the range of 00.00 to 99.99.

Value Type: CFNumber
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemMeteringMode

Metering mode used to acquire the image.

Value Type: CFString
Expected Values: Unknown, Average, CenterWeightedAverage, Spot, MultiSpot, Pattern, Partial
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemOrientation

Orientation of the document contents.

Value Type: CFNumber
Expected Values: 0 (landscape), 1 (portrait)
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemPixelHeight

Height, in pixels, of the contents. For example, the image height or the video frame height.

Value Type: CFNumber
Units: pixels
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemPixelWidth

Width, in pixels, of the contents. For example, the image width or the video frame width.

Value Type: CFNumber
Units: pixels
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemProfileName

Name of the color profile used by the document contents.

Value Type: CFString
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemRedEyeOnOff

Whether red-eye reduction was used to take the picture.

Value Type: CFBoolean

Expected Values: 0 (no red-eye reduction mode or unknown), 1 (red-eye reduction used)
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemResolutionHeightDPI

Resolution height, in DPI, of the item.

Value Type: CFNumber
Units: dots per inch (DPI)
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemResolutionWidthDPI

Resolution width, in DPI, of the item.

Value Type: CFNumber
Units: dots per inch (DPI)
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemWhiteBalance

White balance setting of the camera when the picture was taken.

Value Type: CFNumber
Expected Values: 0 (auto white balance), 1 (manual)
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

Video Metadata Attribute Keys

Metadata attribute keys that are common to video files.

kMDItemAudioBitRate

Bit rate of the audio in the media.

Value Type: CFNumber
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemCodecs

Codecs used to encode/decode the media.

Value Type: Array of CFStrings
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemDeliveryType

Method used to deliver streaming media.

Value Type: CFString
Expected Values: "Fast Start", "RTSP"
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemMediaTypes

Media types present in the content. For example, a QuickTime movie may return:

```
kMDItemMediaTypes = (Sound, Video, "Hinted Video Track", "Hinted Sound Track")
kMDItemMediaTypes = (Sound, Video)
kMDItemMediaTypes = ("MPEG1 Muxed")
```

Value Type: Array of CFStrings
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemStreamable

Whether the content is prepared for streaming.

Value Type: CFBoolean
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemTotalBitRate

Total bit rate, audio and video combined, of the media.

Value Type: CFNumber

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemVideoBitRate

Bit rate of the video in the media.

Value Type: CFNumber

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

Audio Metadata Attribute Keys

Metadata attribute keys that describe an audio file.

kMDItemAppleLoopDescriptors

Specifies multiple pieces of descriptive information about a loop. Besides genre and instrument, files can contain descriptive information that help users in refining searches.

Value Type: Array of CFStrings

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemAppleLoopsKeyFilterType

Specifies key filtering information about a loop. Loops are matched against projects that often differ in a major or minor key. To assist users in identifying loops that will "fit" with their compositions, loops can be tagged with one of the following key filters: "AnyKey", "Minor", "Major", "NeitherKey", or "BothKeys". "AnyKey" means that it fits with anything (whether in a major key, minor key or neither). "Minor" fits with compositions in a minor key. "NeitherKey" doesn't work well with compositions that are in major or minor key. "BothKeys" means it fits with compositions that are in major or minor key.

Value Type: CFString

Expected Values: "AnyKey", "Minor", "Major" , "NeitherKey", "BothKeys"

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemAppleLoopsLoopMode

Specifies how a file should be played. Tagged files can either be loops or non-loops (e.g., a cymbal crash). "Looping" indicates if the file should be treated as a loop. "Non-looping" indicates the file should not be treated as a loop.

Value Type: CFString

Expected Values: "Looping", "Non-looping"

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemAppleLoopsRootKey

Specifies the loop's original key. The key is the root note or tonic for the loop, and does not include the scale type

Value Type: CFString

Expected Values: "C", "C#/Db", "D", "D#/Eb", "E", "F", "F#/Gb", "G", "G#/Ab", "A", "A#/Bb", "B", "NoKey"

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemAudioChannelCount

Number of channels in the audio data contained in the file. This integer value only represents the number of discrete channels of audio data found in the file. It does not indicate any configuration of the data in regards to a user's speaker setup.

Value Type: CFNumber

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemAudioEncodingApplication

Name of the application that encoded the audio of the document.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemAudioSampleRate

Sample rate of the item's audio data. The sample rate is a float value representing hz (audio_frames/second). For example: 44100.0, 22254.54.

Value Type: CFNumber
Units: hz (audio_frames/second)
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemAudioTrackNumber

Track number of a song or composition when it is part of an album.

Value Type: CFNumber
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemComposer

Composer of the song in the audio file.

Value Type: CFString
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemIsGeneralMIDISequence

Whether the MIDI sequence contained in the file is set up for use with a General MIDI device.

Value Type: CFBoolean
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemKeySignature

Musical key of the song in the audio file. For example: "C", "Dm", "F#m", "Bb".

Value Type: CFString
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemLyricist

Lyricist of the song in the audio file.

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemMusicalGenre

Musical genre of the song or composition contained in the audio file. For example: "Jazz", "Pop", "Rock", "Classical".

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemMusicalInstrumentCategory

Specifies the category of an instrument. Files should have an instrument associated with them ("Other Instrument" is provided as a catch-all). For some categories, such as "Keyboards", there are instrument names which provide a more detailed instrument definition, for example "Piano" or "Organ".

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemMusicalInstrumentName

Specifies the name of instrument relative to the instrument category. Files can have an instrument name associated with them if they have certain instrument categories. For example, the "Percussion" category has multiple instruments, including "Conga" and "Bongo".

Value Type: CFString

Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemRecordingDate

Recording date of the song or composition. This is in contrast to `kMDItemContentCreationDate` which, could indicate the creation date of an edited or "mastered" version of the original art.

Value Type: CFDate
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemRecordingYear

Year the item was recorded. For example: 1964, 1995, 1997, or 2003.

Value Type: CFNumber
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemTempo

Tempo of the music in the audio file. A floating point value.

Value Type: CFNumber
Units: Beats per Minute (BPM)
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemTimeSignature

Time signature of the musical composition contained in the audio/MIDI file. For example: "4/4", "7/8".

Value Type: CFString
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

File System Metadata Attribute Keys

Metadata attribute keys that describe the file system attributes for a file. These attributes are available for files on any mounted volume.

kMDItemFSContentChangeDate

Date the file contents last changed.

Value Type: CFDate
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemFSCreationDate

Date that the contents of the file were created.

Value Type: CFDate
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemFSInvisible

Whether the file is invisible.

Value Type: CFBoolean
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemFSIsExtensionHidden

Whether the file extension of the file is hidden.

Value Type: CFBoolean
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemFSLabel

Index of the Finder label of the file. Possible values are 0 through 7.

Value Type: CFNumber
Expected Values: 0 through 7
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h

Availability: Available in Mac OS X v10.4 and later.

kMDItemFSName

File name of the item.

Value Type: CFString
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemFSNodeCount

Number of files in a directory.

Value Type: CFNumber
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemFSOwnerGroupID

Group ID of the owner of the file.

Value Type: CFNumber
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemFSOwnerUserID

User ID of the owner of the file.

Value Type: CFNumber
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemFSSize

Size, in bytes, of the file on disk.

Value Type: CFNumber
Units: bytes

Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

kMDItemPath

Complete path to the file. This value of this attribute can be retrieved, but can't be used in a query or to sort search results. This attribute can't be used as a member of the `valueListAttrs` array parameter for `MDQueryCreate` (page 155) or `MDQueryCreateSubset` (page 155).

Value Type: CFString
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Available in Mac OS X v10.4 and later.

Deprecated Metadata Attribute Keys

Metadata attribute keys that have been deprecated.

kMDItemFSExists

This attribute is deprecated and was never implemented.

Value Type: CFBoolean
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Deprecated in Mac OS X v10.4 and later.

kMDItemFSIsReadable

This attribute is deprecated and was never implemented.

Value Type: CFBoolean
Framework Path: CoreServices/CoreServices.h
Header: MDItem.h
Availability: Deprecated in Mac OS X v10.4 and later.

kMDItemFSIsWriteable

This attribute is deprecated and was never implemented.

Value Type: CFBoolean
Framework Path: CoreServices/CoreServices.h

Header: MDItem.h

Availability: Deprecated in Mac OS X v10.4 and later.

Document Revision History

This table describes the changes to *Core Services Framework Reference*.

Date	Notes
2007-10-31	Added three documents (Folder Manager, Launch Services, and CFStream Socket Additions) and removed one (Device Manager).
2006-07-31	Added information about Identity Services and the Backup Core API.
2006-05-23	First publication of this content as a collection of previously published documents.

Document Revision History

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z