NSFileManager Class Reference

Cocoa > File Management

2008-10-15

Contents

NSFileManager Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Companion guide	Low-Level File Management Programming Topics
Declared in	NSFileManager.h
Related sample code	Core Data HTML Store CoreRecipes MyPhoto Quartz Composer WWDC 2005 TextEdit TextEditPlus

Overview

`NSFileManager` enables you to perform many generic file-system operations and insulates an application from the underlying file system.

Tasks

Getting the Default Manager

`+ defaultManager` (page 10)
    Returns the default `NSFileManager` object for the file system.

Moving an Item

`- movePath:toPath:handler:` (page 34)
    Moves the directory or file specified by a given path to a different location in the file system identified by another path.

- `fileManager:shouldMoveItemAtPath:toPath:` (page 42) *delegate method*
  An `NSFileManager` object sends this message immediately before attempting to move to a given path.
- `moveItemAtPath:toPath:error:` (page 34)
  Moves the directory or file specified by a given path to a different location in the file system identified by another path.
- `fileManager:shouldProceedAfterError:movingItemAtPath:toPath:` (page 45) *delegate method*
  An `NSFileManager` object sends this message if an error occurs during an attempt to move to a given path.

Copying an Item

- `copyPath:toPath:handler:` (page 16)
  Copies the directory or file specified in a given path to a different location in the file system identified by another path.
- `fileManager:shouldCopyItemAtPath:toPath:` (page 41) *delegate method*
  An `NSFileManager` object sends this message immediately before attempting to copy to a given path.
- `copyItemAtPath:toPath:error:` (page 15)
  Copies the directory or file specified in a given path to a different location in the file system identified by another path.
- `fileManager:shouldProceedAfterError:copyingItemAtPath:toPath:` (page 44) *delegate method*
  An `NSFileManager` object sends this message if an error occurs during an attempt to copy to a given path.

Removing an Item

- `removeFileAtPath:handler:` (page 36)
  Deletes the file, link, or directory (including, recursively, all subdirectories, files, and links in the directory) identified by a given path.
- `fileManager:shouldRemoveItemAtPath:` (page 46) *delegate method*
  An `NSFileManager` object sends this message immediately before attempting to delete an item at a given path.
- `removeItemAtPath:error:` (page 37)
  Deletes the file, link, or directory (including, recursively, all subdirectories, files, and links in the directory) identified by a given path.
- `fileManager:shouldProceedAfterError:removingItemAtPath:` (page 46) *delegate method*
  An `NSFileManager` object sends this message if an error occurs during an attempt to delete a given path.

Creating an Item

- `createDirectoryAtPath:attributes:` (page 17)
  Creates a directory (without contents) at a given path with given attributes.

- `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 18)
  Creates a directory with given attributes at a specified path.
- `createFileAtPath:contents:attributes:` (page 19)
  Creates a file at a given path that has given attributes and contents.

Linking an Item

- `linkPath:toPath:handler:` (page 32)
  Creates a link from a source to a destination.
- `fileManager:shouldLinkItemAtPath:toPath:` (page 42) *delegate method*
  An `NSFileManager` object sends this message immediately before attempting to link to a given path.
- `linkItemAtPath:toPath:error:` (page 31)
  Creates a link from a source to a destination.
- `fileManager:shouldProceedAfterError:linkingItemAtPath:toPath:` (page 45) *delegate method*
  An `NSFileManager` object sends this message if an error occurs during an attempt to link to a given path.

Symbolic-Link Operations

- `createSymbolicLinkAtPath:pathContent:` (page 20)
  Creates a symbolic link identified by a given path that refers to a given location.
- `createSymbolicLinkAtPath:withDestinationPath:error:` (page 20)
  Creates a symbolic link identified by a given path that refers to a given location.
- `pathContentOfSymbolicLinkAtPath:` (page 36)
  Returns the path of the directory or file that a symbolic link at a given path refers to.
- `destinationOfSymbolicLinkAtPath:error:` (page 22)
  Returns an `NSString` object containing the path of the item pointed at by the symlink specified by a given path.

Handling File Operations

The methods described in this section are methods to be implemented by the callback handler passed to several methods of `NSFileManager`.

- `fileManager:shouldProceedAfterError:` (page 43) *delegate method*
  An `NSFileManager` object sends this message to its handler for each error it encounters when copying, moving, removing, or linking files or directories.
- `fileManager:willProcessPath:` (page 47) *delegate method*
  An `NSFileManager` object sends this message to a handler immediately before attempting to move, copy, rename, or delete, or before attempting to link to a given path.

Getting and Comparing File Contents

- `contentsAtPath:` (page 13)
  Returns as an `NSData` object the contents of the file at at given path.
- `contentsEqualAtPath:andPath:` (page 14)
  Returns a Boolean value that indicates whether the files or directories in specified paths have the same contents.

Discovering Directory Contents

- `directoryContentsAtPath:` (page 22)
  Returns an array of `NSString` objects identifying the directories and files (including symbolic links) contained in a given directory.
- `contentsOfDirectoryAtPath:error:` (page 14)
  Returns an array of `NSString` objects identifying the directories and files (including symbolic links) contained in a given directory.
- `enumeratorAtPath:` (page 24)
  Creates and returns an `NSDirectoryEnumerator` object that enumerates the contents of the directory at a given path.
- `subpathsAtPath:` (page 39)
  Returns an array that contains (as `NSString` objects) the contents of the directory identified by a given path.
- `subpathsOfDirectoryAtPath:error:` (page 40)
  Returns an array that contains the filenames of the items in the directory specified by a given path and all its subdirectories recursively.

Determining Access to Files

- `fileExistsAtPath:` (page 26)
  Returns a Boolean value that indicates whether a file or directory exists at a specified path.
- `fileExistsAtPath:isDirectory:` (page 27)
  Returns a Boolean value that indicates whether a file or directory exists at a specified path.
- `isReadableFileAtPath:` (page 30)
  Returns a Boolean value that indicates whether the invoking object appears able to read a specified file.
- `isWritableFileAtPath:` (page 31)
  Returns a Boolean value that indicates whether the invoking object appears able to write to a specified file.
- `isExecutableFileAtPath:` (page 30)
  Returns a Boolean value that indicates whether the operating system appears able to execute a specified file.
- `isDeletableFileAtPath:` (page 29)
  Returns a Boolean value that indicates whether the invoking object appears able to delete a specified file.

Getting and Setting Attributes

- `componentsToDisplayForPath:` (page 13)
  Returns an array of `NSString` objects representing the user-visible components of a given path.
- `displayNameAtPath:` (page 23)
  Returns the name of the file or directory at a given path in a localized form appropriate for presentation to the user.
- `fileAttributesAtPath:traverseLink:` (page 25)
  Returns a dictionary that describes the POSIX attributes of the file specified at a given.
- `attributesOfItemAtPath:error:` (page 11)
  An `NSDictionary` object containing the attributes of the item at a given path.
- `fileSystemAttributesAtPath:` (page 28)
  Returns a dictionary that describes the attributes of the mounted file system on which a given path resides.
- `attributesOfFileSystemForPath:error:` (page 10)
  Returns a dictionary that describes the attributes of the mounted file system on which a given path resides.
- `changeFileAttributes:atPath:` (page 12)
  Changes the attributes of a given file or directory.
- `setAttributes:ofItemAtPath:error:` (page 38)
  Sets the attributes of a given file or directory.

Getting Representations of File Paths

- `fileSystemRepresentationWithPath:` (page 29)
  Returns a C-string representation of a given path that properly encodes Unicode strings for use by the file system.
- `stringWithFileSystemRepresentation:length:` (page 39)
  Returns an `NSString` object converted from the C-string representation of a pathname in the current file system.

Managing the Delegate

- `setDelegate:` (page 39)
  Sets the delegate for the receiver.
- `delegate` (page 21)
  Returns the delegate for the receiver.

Managing the Current Directory

- `changeCurrentDirectoryPath:` (page 11)
  Changes the path of the current directory for the current process to a given path.
- `currentDirectoryPath` (page 21)
  Returns the path of the program's current directory.

Class Methods

defaultManager

Returns the default `NSFileManager` object for the file system.

```
+ (NSFileManager *)defaultManager
```

Return Value
The default `NSFileManager` object for the file system.

Discussion
You invoke all `NSFileManager` instance methods with this object as the receiver.

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

Related Sample Code
CIVideoDemoGL

Core Data HTML Store

CoreRecipes

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In
`NSFileManager.h`

Instance Methods

attributesOfFileSystemForPath:error:

Returns a dictionary that describes the attributes of the mounted file system on which a given path resides.

```
- (NSDictionary *)attributesOfFileSystemForPath:(NSString *)path error:(NSError
    **)error
```

Parameters

path
Any pathname within the mounted file system.

error
If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
An `NSDictionary` object that describes the attributes of the mounted file system on which *path* resides. See "File-System Attribute Keys" (page 52) for a description of the keys available in the dictionary.

Discussion
This method does not traverse an initial symbolic link.

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

See Also
`- fileSystemAttributesAtPath:` (page 28)
`- fileAttributesAtPath:traverseLink:` (page 25)
`- changeFileAttributes:atPath:` (page 12)

Declared In
`NSFileManager.h`

attributesOfItemAtPath:error:

An `NSDictionary` object containing the attributes of the item at a given path.

```
- (NSDictionary *)attributesOfItemAtPath:(NSString *)path error:(NSError **)error
```

Parameters

path
The path of a file or directory.

error
If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
An `NSDictionary` object that describes the attributes (file, directory, symlink, and so on) of the file specified by *path*. The keys in the dictionary are described in "`File Attribute Keys`" (page 48).

Discussion
This method does not traverse an initial symbolic link.

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

See Also
`- fileAttributesAtPath:traverseLink:` (page 25)
`- changeFileAttributes:atPath:` (page 12)

Declared In
`NSFileManager.h`

changeCurrentDirectoryPath:

Changes the path of the current directory for the current process to a given path.

```
- (BOOL)changeCurrentDirectoryPath:(NSString *)path
```

Parameters

path
The path of the directory to which to change.

Return Value
`YES` if successful, otherwise `NO`.

Discussion

All relative pathnames refer implicitly to the current working directory. The current working directory is stored per process.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `currentDirectoryPath` (page 21)
- `fileExistsAtPath:isDirectory:` (page 27)
- `directoryContentsAtPath:` (page 22)
- `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 18)
- `createDirectoryAtPath:attributes:` (page 17)

Declared In

`NSFileManager.h`

changeFileAttributes:atPath:

Changes the attributes of a given file or directory.

```
- (BOOL)changeFileAttributes:(NSDictionary *)attributes  atPath:(NSString *)path
```

Parameters

`attributes`

A dictionary containing as keys the attributes to set for *`path`* and as values the corresponding value for the attribute. You can set following: `NSFileBusy`, `NSFileCreationDate`, `NSFileExtensionHidden`, `NSFileGroupOwnerAccountID`, `NSFileGroupOwnerAccountName`, `NSFileHFSCreatorCode`, `NSFileHFSTypeCode`, `NSFileImmutable`, `NSFileModificationDate`, `NSFileOwnerAccountID`, `NSFileOwnerAccountName`, `NSFilePosixPermissions`. You can change single attributes or any combination of attributes; you need not specify keys for all attributes.

For the `NSFilePosixPermissions` value, specify a file mode from the OR'd permission bit masks defined in `sys/stat.h`. See the man page for the `chmod` function (`man 2 chmod`) for an explanation.

`path`

A path to a file or directory.

Return Value

`YES` if *all* changes succeed. If any change fails, returns `NO`, but it is undefined whether any changes actually occurred.

Discussion

As in the POSIX standard, the application either must own the file or directory or must be running as superuser for attribute changes to take effect. The method attempts to make all changes specified in attributes and ignores any rejection of an attempted modification.

The `NSFilePosixPermissions` value must be initialized with the code representing the POSIX file-permissions bit pattern. `NSFileHFSCreatorCode` and `NSFileHFSTypeCode` will only be heeded when *`path`* specifies a file.

Special Considerations

On Mac OS X v10.5 and later, use `setAttributes:ofItemAtPath:error:` (page 38) instead.

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

See Also
- `fileAttributesAtPath:traverseLink:` (page 25)
- `setAttributes:ofItemAtPath:error:` (page 38)

Related Sample Code
File Wrappers with Core Data Documents
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSFileManager.h`

componentsToDisplayForPath:

Returns an array of `NSString` objects representing the user-visible components of a given path.

```
- (NSArray *)componentsToDisplayForPath:(NSString *)path
```

Parameters

`path`
A pathname.

Return Value
An array of `NSString` objects representing the user-visible (for the Finder, Open and Save panels, and so on) components of *`path`*.

Discussion
These components cannot be used for path operations and are only suitable for display to the user.

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

Related Sample Code
QTAudioExtractionPanel

Declared In
`NSFileManager.h`

contentsAtPath:

Returns as an `NSData` object the contents of the file at at given path.

```
- (NSData *)contentsAtPath:(NSString *)path
```

Parameters

`path`
The path of a file.

Return Value
The contents of the file specified by *`path`* as an `NSData` object. If *`path`* specifies a directory, or if some other error occurs, returns `nil`.

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

See Also
`– contentsEqualAtPath:andPath:` (page 14)
`– createFileAtPath:contents:attributes:` (page 19)

Declared In
`NSFileManager.h`

contentsEqualAtPath:andPath:

Returns a Boolean value that indicates whether the files or directories in specified paths have the same contents.

```
- (BOOL)contentsEqualAtPath:(NSString *)path1  andPath:(NSString *)path2
```

Parameters

`path1`
: The path of a file or directory to compare with the contents of *`path2`*.

`path2`
: The path of a file or directory to compare with the contents of *`path1`*.

Return Value
`YES` if file or directory specified in *`path1`* has the same contents as that specified in *`path2`*, otherwise `NO`.

Discussion
If *`path1`* and *`path2`* are directories, the contents are the list of files and subdirectories each contains—contents of subdirectories are also compared. For files, this method checks to see if they're the same file, then compares their size, and finally compares their contents. This method does not traverse symbolic links, but compares the links themselves.

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

See Also
`– contentsAtPath:` (page 13)

Declared In
`NSFileManager.h`

contentsOfDirectoryAtPath:error:

Returns an array of `NSString` objects identifying the directories and files (including symbolic links) contained in a given directory.

```
- (NSArray *)contentsOfDirectoryAtPath:(NSString *)path error:(NSError **)error
```

Parameters

path

A path to a directory.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value

An array of `NSString` objects identifying the directories and files (including symbolic links) contained in *path*. Returns `nil` if the directory exists but has no contents. Returns `nil` if the directory specified at *path* does not exist or there is some other error accessing it.

Discussion

The search is shallow and therefore does not return the contents of any subdirectories. This returned array does not contain strings for the current directory ("."), parent directory (".."), or resource forks (begin with "._") and does not traverse symbolic links.

Availability

Available in Mac OS X v10.5 and later.

See Also

`– directoryContentsAtPath:` (page 22)
`– currentDirectoryPath` (page 21)
`– fileExistsAtPath:isDirectory:` (page 27)
`– enumeratorAtPath:` (page 24)
`– subpathsAtPath:` (page 39)

Declared In

`NSFileManager.h`

copyItemAtPath:toPath:error:

Copies the directory or file specified in a given path to a different location in the file system identified by another path.

```
- (BOOL)copyItemAtPath:(NSString *)srcPath toPath:(NSString *)dstPath error:(NSError
    **)error
```

Parameters

srcPath

The path of a file or directory.

dstPath

The path of a file or directory.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value

`YES` if the operation was successful, otherwise `NO`.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `fileManager:shouldCopyItemAtPath:toPath:` (page 41)
- `fileManager:shouldProceedAfterError:copyingItemAtPath:toPath:` (page 44)
- `linkItemAtPath:toPath:error:` (page 31)
- `moveItemAtPath:toPath:error:` (page 34)
- `removeItemAtPath:error:` (page 37)
- `copyPath:toPath:handler:` (page 16)

Declared In

`NSFileManager.h`

copyPath:toPath:handler:

Copies the directory or file specified in a given path to a different location in the file system identified by another path.

```
- (BOOL)copyPath:(NSString *)source  toPath:(NSString *)destination
    handler:(id)handler
```

Parameters

`source`

The location of the source file.

`destination`

The location to which to copy the file specified by *`source`*.

`handler`

An object that responds to the callback messages `fileManager:willProcessPath:` (page 47) and `fileManager:shouldProceedAfterError:` (page 43). You can specify `nil` for *`handler`*; if you do so and an error occurs, the method automatically returns `NO`.

Return Value

`YES` if the copy operation is successful. If the operation is not successful, but the callback handler of `fileManager:shouldProceedAfterError:` (page 43) returns `YES`, `copyPath:toPath:handler:` also returns `YES`. Otherwise this method returns `NO`. The method also attempts to make the attributes of the directory or file at *`destination`* identical to *`source`*, but ignores any failure at this attempt.

Discussion

If *`source`* is a file, the method creates a file at *`destination`* that holds the exact contents of the original file (this includes BSD special files). If *`source`* is a directory, the method creates a new directory at *`destination`* and recursively populates it with duplicates of the files and directories contained in *`source`*, preserving all links. The file specified in *`source`* must exist, while *`destination`* must not exist prior to the operation. When a file is being copied, the destination path must end in a filename—there is no implicit adoption of the source filename. Symbolic links are not traversed but are themselves copied. File or directory attributes—that is, metadata such as owner and group numbers, file permissions, and modification date—are also copied.

The handler callback mechanism is similar to delegation. `NSFileManager` sends `fileManager:willProcessPath:` (page 47) when it begins a copy, move, remove, or link operation. It sends `fileManager:shouldProceedAfterError:` (page 43) when it encounters any error in processing.

This code fragment verifies that the file to be copied exists and then copies that file to the user's `~/Library/Reports` directory:

```
NSString *source = @"/tmp/quarterly_report.rtf";
NSString *destination = [[[NSHomeDirectory()
        stringByAppendingPathComponent:@"Library"]
        stringByAppendingPathComponent:@"Reports"]
        stringByAppendingPathComponent:@"new_quarterly_report.rtf"];
NSFileManager *fileManager = [NSFileManager defaultManager];

if ([fileManager fileExistsAtPath:source]) {
    [fileManager copyPath:source toPath:destination handler:nil];
}
```

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

See Also
- `linkPath:toPath:handler:` (page 32)
- `movePath:toPath:handler:` (page 34)
- `fileManager:shouldProceedAfterError:` (page 43)
- `removeFileAtPath:handler:` (page 36)
- `fileManager:willProcessPath:` (page 47)

Related Sample Code
Core Data HTML Store

Declared In
`NSFileManager.h`

createDirectoryAtPath:attributes:

Creates a directory (without contents) at a given path with given attributes.

```
- (BOOL)createDirectoryAtPath:(NSString *)path attributes:(NSDictionary *)attributes
```

Parameters

path
The path at which to create the new directory. The directory to be created must not yet exist, but its parent directory must exist.

attributes
The file attributes for the new directory. The attributes you can set are owner and group numbers, file permissions, and modification date. If you specify `nil` for *attributes*, default values for these attributes are set (particularly write access for the creator and read access for others). The "Constants" (page 48) section lists the global constants used as keys in the *attributes* dictionary. Some of the keys, such as `NSFileHFSCreatorCode` and `NSFileHFSTypeCode`, do not apply to directories.

Return Value
`YES` if the operation was successful, otherwise `NO`.

Special Considerations
On Mac OS X v10.5 and later, use
`createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 18) instead.

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

See Also
- `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 18)
- `changeCurrentDirectoryPath:` (page 11)
- `changeFileAttributes:atPath:` (page 12)
- `createFileAtPath:contents:attributes:` (page 19)
- `currentDirectoryPath` (page 21)

Related Sample Code
Core Data HTML Store
CoreRecipes
GridCalendar
MyPhoto
SpotlightFortunes

Declared In
`NSFileManager.h`

createDirectoryAtPath:withIntermediateDirectories:attributes:error:

Creates a directory with given attributes at a specified path.

```
- (BOOL)createDirectoryAtPath:(NSString *)path
    withIntermediateDirectories:(BOOL)createIntermediates attributes:(NSDictionary
     *)attributes error:(NSError **)error
```

Parameters

path
: The path at which to create the new directory. The directory to be created must not yet exist.

createIntermediates
: If `YES`, then the method will also create any necessary intermediate directories; if `NO`, then the method will fail if any parent of the directory to be created does not exist.

attributes
: The file attributes for the new directory. The attributes you can set are owner and group numbers, file permissions, and modification date. If you specify `nil` for *attributes*, the directory is created according to the umask of the process. The "Constants" (page 48) section lists the global constants used as keys in the *attributes* dictionary. Some of the keys, such as `NSFileHFSCreatorCode` and `NSFileHFSTypeCode`, do not apply to directories.

error
: If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
`YES` if the operation was successful, otherwise `NO`.

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

See Also

- `createDirectoryAtPath:attributes:` (page 17)
- `changeCurrentDirectoryPath:` (page 11)
- `setAttributes:ofItemAtPath:error:` (page 38)
- `createFileAtPath:contents:attributes:` (page 19)
- `currentDirectoryPath` (page 21)

Declared In

`NSFileManager.h`

createFileAtPath:contents:attributes:

Creates a file at a given path that has given attributes and contents.

```
- (BOOL)createFileAtPath:(NSString *)path   contents:(NSData *)contents
    attributes:(NSDictionary *)attributes
```

Parameters

path

The path for the new file.

contents

The contents for the new file.

attributes

A dictionary that describes the attributes of the new file. The file attributes you can set are owner and group numbers, file permissions, and modification date. "File Attribute Keys" (page 48) lists the global constants used as keys in the *attributes* dictionary. If you specify `nil` for *attributes*, the file is given a default set of attributes.

Return Value

`YES` if the operation was successful, otherwise `NO`.

Discussion

If a file already exists at *path*, then if the file can be overwritten (subject to user privileges) it will be.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `contentsAtPath:` (page 13)
- `changeFileAttributes:atPath:` (page 12)
- `setAttributes:ofItemAtPath:error:` (page 38)
- `fileAttributesAtPath:traverseLink:` (page 25)
- `attributesOfItemAtPath:error:` (page 11)

Related Sample Code

Core Data HTML Store

CustomAtomicStoreSubclass

TimelineToTC

Declared In

`NSFileManager.h`

createSymbolicLinkAtPath:pathContent:

Creates a symbolic link identified by a given path that refers to a given location.

```
- (BOOL)createSymbolicLinkAtPath:(NSString *)path pathContent:(NSString *)otherPath
```

Parameters

path

The path for a symbolic link.

otherPath

The path to which *path* should refer.

Return Value

`YES` if the operation is successful, otherwise `NO`. Returns `NO` if a file, directory, or symbolic link identical to *path* already exists.

Discussion

Creates a symbolic link identified by *path* that refers to the location *otherPath* in the file system.

Special Considerations

On Mac OS X v10.5 and later, use `createSymbolicLinkAtPath:withDestinationPath:error:` (page 20) instead.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `createSymbolicLinkAtPath:withDestinationPath:error:` (page 20)
- `pathContentOfSymbolicLinkAtPath:` (page 36)
- `linkPath:toPath:handler:` (page 32)

Declared In

`NSFileManager.h`

createSymbolicLinkAtPath:withDestinationPath:error:

Creates a symbolic link identified by a given path that refers to a given location.

```
- (BOOL)createSymbolicLinkAtPath:(NSString *)path withDestinationPath:(NSString
    *)destPath error:(NSError **)error
```

Parameters

path

The path for a symbolic link.

destPath

The path to which *path* should refer.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value

`YES` if the operation is successful, otherwise `NO`. Returns `NO` if a file, directory, or symbolic link identical to *path* already exists.

Discussion
Creates a symbolic link identified by *`path`* that refers to the location *`destPath`* in the file system.

This method does not traverse an initial symlink.

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

See Also
- `createSymbolicLinkAtPath:pathContent:` (page 20)
- `pathContentOfSymbolicLinkAtPath:` (page 36)
- `linkPath:toPath:handler:` (page 32)

Declared In
`NSFileManager.h`

currentDirectoryPath

Returns the path of the program's current directory.

```
- (NSString *)currentDirectoryPath
```

Return Value
The path of the program's current directory. If the program's current working directory isn't accessible, returns `nil`.

Discussion
The string returned by this method is initialized to the current working directory; you can change the working directory by invoking `changeCurrentDirectoryPath:` (page 11).

Relative pathnames refer implicitly to the current directory. For example, if the current directory is `/tmp`, and the relative pathname `reports/info.txt` is specified, the resulting full pathname is `/tmp/reports/info.txt`.

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

See Also
- `changeCurrentDirectoryPath:` (page 11)
- `createDirectoryAtPath:attributes:` (page 17)
- `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 18)

Declared In
`NSFileManager.h`

delegate

Returns the delegate for the receiver.

```
- (id)delegate
```

Return Value
The delegate for the receiver.

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

Declared In
`NSFileManager.h`

destinationOfSymbolicLinkAtPath:error:

Returns an `NSString` object containing the path of the item pointed at by the symlink specified by a given path.

```
- (NSString *)destinationOfSymbolicLinkAtPath:(NSString *)path error:(NSError
    **)error
```

Parameters
path

The path of a file or directory.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
An `NSString` object containing the path of the directory or file to which the symbolic link *path* refers, or `nil` upon failure. If the symbolic link is specified as a relative path, that relative path is returned.

Discussion
This method does not traverse an initial symlink.

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

See Also
`- pathContentOfSymbolicLinkAtPath:` (page 36)
`- createSymbolicLinkAtPath:withDestinationPath:error:` (page 20)

Declared In
`NSFileManager.h`

directoryContentsAtPath:

Returns an array of `NSString` objects identifying the directories and files (including symbolic links) contained in a given directory.

```
- (NSArray *)directoryContentsAtPath:(NSString *)path
```

Parameters
path

A path to a directory.

Return Value
An array of `NSString` objects identifying the directories and files (including symbolic links) contained in *path*. Returns an empty array if the directory exists but has no contents. Returns `nil` if the directory specified at *path* does not exist or there is some other error accessing it.

Discussion

The search is shallow and therefore does not return the contents of any subdirectories. This returned array does not contain strings for the current directory ("."), parent directory (".."), or resource forks (begin with "._") and does not traverse symbolic links.

Special Considerations

On Mac OS X v10.5 and later, use `contentsOfDirectoryAtPath:error:` (page 14) instead.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `contentsOfDirectoryAtPath:error:` (page 14)
- `currentDirectoryPath` (page 21)
- `fileExistsAtPath:isDirectory:` (page 27)
- `enumeratorAtPath:` (page 24)
- `subpathsAtPath:` (page 39)

Related Sample Code

ColorSyncDevices-Cocoa

IKSlideshowDemo

LSMSmartCategorizer

ThreadsImporter

ThreadsImportMovie

Declared In

`NSFileManager.h`

displayNameAtPath:

Returns the name of the file or directory at a given path in a localized form appropriate for presentation to the user.

```
- (NSString *)displayNameAtPath:(NSString *)path
```

Parameters

`path`

The path of a file or directory.

Return Value

The name of the file or directory at *`path`* in a localized form appropriate for presentation to the user. If there is no file or directory at *`path`*, or if an error occurs, returns `[path lastPathComponent]`.

Discussion

The returned value is localized where appropriate. For example, if you have selected French as your preferred language, the following code fragment logs "Bibliothèque":

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSLibraryDirectory,
NSUserDomainMask, YES);
if ([paths count] > 0)
{
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSFileManager *fileManager = [NSFileManager defaultManager];
    NSString *displayNameAtPath = [fileManager
displayNameAtPath:documentsDirectory];
    NSLog(@"%@", displayNameAtPath);
}
```

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

See Also
`- lastPathComponent` (`NSString`)

Related Sample Code
AlbumToSlideshow

AutomatorHandsOn

DeskPictAppDockMenu

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In
`NSFileManager.h`

enumeratorAtPath:

Creates and returns an `NSDirectoryEnumerator` object that enumerates the contents of the directory at a given path.

```
- (NSDirectoryEnumerator *)enumeratorAtPath:(NSString *)path
```

Parameters

path
: The path of the directory to enumerate.

Return Value
An `NSDirectoryEnumerator` object that enumerates the contents of the directory at *path*. If *path* is a symbolic link, this method evaluates the link and returns an enumerator for the file or directory the link points to. If the link cannot be evaluated, the method returns `nil`.

If *path* is a filename, the method returns an enumerator object that enumerates no files—the first call to `nextObject` will return `nil`.

Discussion
Because the enumeration is deep—that is, it lists the contents of all subdirectories—this enumerator object is useful for performing actions that involve large file-system subtrees. If the method is passed a directory on which another file system is mounted (a mount point), it traverses the mount point. This method does not resolve symbolic links encountered in the traversal process, nor does it recurse through them if they point to a directory.

This code fragment enumerates the subdirectories and files under a user's `Documents` directory and processes all files with an extension of `.doc`:

```
NSString *file;
NSString *docsDir = [NSHomeDirectory() stringByAppendingPathComponent:
@"Documents"];
NSDirectoryEnumerator *dirEnum =
    [[NSFileManager defaultManager] enumeratorAtPath:docsDir];

while (file = [dirEnum nextObject]) {
    if ([[file pathExtension] isEqualToString: @"doc"]) {
        [self scanDocument: [docsDir stringByAppendingPathComponent:file]];
    }
}
```

The `NSDirectoryEnumerator` class has methods for obtaining the attributes of the existing path and of the parent directory and for skipping descendants of the existing path.

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

See Also
- `currentDirectoryPath` (page 21)
- `fileAttributesAtPath:traverseLink:` (page 25)
- `directoryContentsAtPath:` (page 22)
- `subpathsAtPath:` (page 39)

Related Sample Code
BundleLoader
DeskPictAppDockMenu
NSOperationSample

Declared In
`NSFileManager.h`

fileAttributesAtPath:traverseLink:

Returns a dictionary that describes the POSIX attributes of the file specified at a given.

```
- (NSDictionary *)fileAttributesAtPath:(NSString *)path traverseLink:(BOOL)flag
```

Parameters

`path`
A file path.

`flag`
If *`path`* is not a symbolic link, this parameter has no effect. If *`path`* is a symbolic link, then:

- If `YES` the attributes of the linked-to file are returned, or if the link points to a nonexistent file the method returns `nil`.
- If `NO`, the attributes of the symbolic link are returned.

Return Value
An `NSDictionary` object that describes the POSIX attributes of the file specified at *`path`*. The keys in the dictionary are described in "`File Attribute Keys`" (page 48). If there is no item at *`path`*, returns `nil`.

Discussion
This code example gets several attributes of a file and logs them.

```
NSFileManager *fileManager = [NSFileManager defaultManager];
NSString *path = @"/tmp/List";
```

```
NSDictionary *fileAttributes = [fileManager fileAttributesAtPath:path
traverseLink:YES];

if (fileAttributes != nil) {
    NSNumber *fileSize;
    NSString *fileOwner;
    NSDate *fileModDate;
    if (fileSize = [fileAttributes objectForKey:NSFileSize]) {
        NSLog(@"File size: %qi\n", [fileSize unsignedLongLongValue]);
    }
    if (fileOwner = [fileAttributes objectForKey:NSFileOwnerAccountName]) {
        NSLog(@"Owner: %@\n", fileOwner);
    }
    if (fileModDate = [fileAttributes objectForKey:NSFileModificationDate]) {
        NSLog(@"Modification date: %@\n", fileModDate);
    }
}
else {
    NSLog(@"Path (%@) is invalid.", path);
}
```

As a convenience, `NSDictionary` provides a set of methods (declared as a category in `NSFileManager.h`) for quickly and efficiently obtaining attribute information from the returned dictionary: `fileGroupOwnerAccountName`, `fileModificationDate`, `fileOwnerAccountName`, `filePosixPermissions`, `fileSize`, `fileSystemFileNumber`, `fileSystemNumber`, and `fileType`. For example, you could rewrite the file modification statement in the code example above as:

```
if (fileModDate = [fileAttributes fileModificationDate])
    NSLog(@"Modification date: %@\n", fileModDate);
```

Special Considerations

On Mac OS X v10.5 and later, use `attributesOfItemAtPath:error:` (page 11) instead.

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

See Also
`- attributesOfItemAtPath:error:` (page 11)
`- changeFileAttributes:atPath:` (page 12)

Related Sample Code
AudioBurn
DeskPictAppDockMenu
Quartz Composer WWDC 2005 TextEdit
TextEditPlus
ThreadsImportMovie

Declared In
`NSFileManager.h`

fileExistsAtPath:

Returns a Boolean value that indicates whether a file or directory exists at a specified path.

```
- (BOOL)fileExistsAtPath:(NSString *)path
```

Parameters

path

The path of a file or directory. If `path` begins with a tilde (~), it must first be expanded with `stringByExpandingTildeInPath`, or this method will return `NO`.

Return Value

`YES` if a file specified in `path` exists, otherwise `NO`. If the final element in `path` specifies a symbolic link, this method traverses the link and returns `YES` or `NO` based on the existence of the file at the link destination.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- fileExistsAtPath:isDirectory:` (page 27)

Related Sample Code

CoreRecipes

QTKitCreateMovie

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In

`NSFileManager.h`

fileExistsAtPath:isDirectory:

Returns a Boolean value that indicates whether a file or directory exists at a specified path.

```
- (BOOL)fileExistsAtPath:(NSString *)path   isDirectory:(BOOL *)isDirectory
```

Parameters

path

The path of a file or directory. If `path` begins with a tilde (~), it must first be expanded with `stringByExpandingTildeInPath`, or this method will return `NO`.

isDirectory

Upon return, contains `YES` if `path` is a directory or if the final path element is a symbolic link that points to a directory, otherwise contains `NO`. If `path` doesn't exist, the return value is undefined. Pass `NULL` if you do not need this information.

Return Value

`YES` if there is a file or directory at `path`, otherwise `NO`. If the final element in `path` specifies a symbolic link, this method traverses the link and returns `YES` or `NO` based on the existence of the file or directory at the link destination.

Discussion

If you need to further determine if `path` is a package, use the `NSWorkspace` method `isFilePackageAtPath:`.

This example gets an array that identifies the fonts in the user's fonts directory:

```
NSArray *subpaths;
BOOL isDir;
```

```
NSArray *paths = NSSearchPathForDirectoriesInDomains
                     (NSLibraryDirectory, NSUserDomainMask, YES);

if ([paths count] == 1) {

    NSFileManager *fileManager = [NSFileManager defaultManager];
    NSString *fontPath = [[paths objectAtIndex:0]
stringByAppendingPathComponent:@"Fonts"];

    if ([fileManager fileExistsAtPath:fontPath isDirectory:&isDir] && isDir) {
        subpaths = [fileManager subpathsAtPath:fontPath];
// ...
```

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

See Also
`- fileExistsAtPath:` (page 26)

Related Sample Code
ImageBrowser
LSMSmartCategorizer
QTKitCreateMovie
QTKitImport
QTKitPlayer

Declared In
`NSFileManager.h`

fileSystemAttributesAtPath:

Returns a dictionary that describes the attributes of the mounted file system on which a given path resides.

```
- (NSDictionary *)fileSystemAttributesAtPath:(NSString *)path
```

Parameters

`path`
Any pathname within the mounted file system.

Return Value
An `NSDictionary` object that describes the attributes of the mounted file system on which *`path`* resides. See "File-System Attribute Keys" (page 52) for a description of the keys available in the dictionary.

Discussion
The following code example checks to see if there's sufficient space on the file system before adding a new file to it:

```
NSData *contents = [myImage TIFFRepresentation];
NSFileManager *fileManager = [NSFileManager defaultManager];
NSString *path = ...;
NSString *fileName = ...;
NSDictionary *fsAttributes =
        [fileManager fileSystemAttributesAtPath:path];
if ([[fsAttributes objectForKey:NSFileSystemFreeSize] unsignedLongLongValue]
>
```

```
        [contents length])
    [fileManager createFileAtPath:[path stringByAppendingPathComponent:fileName]
            contents:contents attributes:nil];
```

Special Considerations

On Mac OS X v10.5 and later, use `attributesOfFileSystemForPath:error:` (page 10) instead.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `attributesOfFileSystemForPath:error:` (page 10)
- `fileAttributesAtPath:traverseLink:` (page 25)
- `changeFileAttributes:atPath:` (page 12)

Declared In

`NSFileManager.h`

fileSystemRepresentationWithPath:

Returns a C-string representation of a given path that properly encodes Unicode strings for use by the file system.

```
- (const char *)fileSystemRepresentationWithPath:(NSString *)path
```

Parameters

path

A file path.

Return Value

A C-string representation of *path* that properly encodes Unicode strings for use by the file system.

Discussion

If you need the C string beyond the scope of your autorelease pool, you must copy it. This method raises an exception upon error. Use this method if your code calls system routines that expect C-string path arguments.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `stringWithFileSystemRepresentation:length:` (page 39)

Declared In

`NSFileManager.h`

isDeletableFileAtPath:

Returns a Boolean value that indicates whether the invoking object appears able to delete a specified file.

```
- (BOOL)isDeletableFileAtPath:(NSString *)path
```

Parameters

path

A file path.

Return Value

`YES` if the invoking object appears able to delete the file specified in *path*, otherwise `NO`. If the file at *path* does not exist, this method returns `NO`.

Discussion

For a directory or file to be able to be deleted, either the parent directory of *path* must be writable or its owner must be the same as the owner of the application process. If *path* is a directory, every item contained in *path* must be able to be deleted.

This method does not traverse symbolic links.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSFileManager.h`

isExecutableFileAtPath:

Returns a Boolean value that indicates whether the operating system appears able to execute a specified file.

```
- (BOOL)isExecutableFileAtPath:(NSString *)path
```

Parameters

path

A file path.

Return Value

`YES` if the operating system appears able to execute the file specified in *path*, otherwise `NO`. If the file at *path* does not exist, this method returns `NO`.

Discussion

This method traverses symbolic links. This method uses the real user ID and group ID, as opposed to the effective user and group IDs, to determine if the file is executable.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSFileManager.h`

isReadableFileAtPath:

Returns a Boolean value that indicates whether the invoking object appears able to read a specified file.

```
- (BOOL)isReadableFileAtPath:(NSString *)path
```

Parameters

path

A file path.

Return Value

`YES` if the invoking object appears able to read the file specified in *path*, otherwise `NO`. If the file at *path* does not exist, this method returns `NO`.

Discussion

This method traverses symbolic links. This method uses the real user ID and group ID, as opposed to the effective user and group IDs, to determine if the file is readable.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTQuartzPlayer

Declared In

`NSFileManager.h`

isWritableFileAtPath:

Returns a Boolean value that indicates whether the invoking object appears able to write to a specified file.

```
- (BOOL)isWritableFileAtPath:(NSString *)path
```

Parameters

path

A file path.

Return Value

`YES` if the invoking object appears able to write to the file specified in *path*, otherwise `NO`. If the file at *path* does not exist, this method returns `NO`.

Discussion

This method traverses symbolic links. This method uses the real user ID and group ID, as opposed to the effective user and group IDs, to determine if the file is writable.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSFileManager.h`

linkItemAtPath:toPath:error:

Creates a link from a source to a destination.

```
- (BOOL)linkItemAtPath:(NSString *)srcPath toPath:(NSString *)dstPath error:(NSError
    **)error
```

Parameters

`srcPath`

A path that identifies a source file.

The file or link specified by *`srcPath`* must exist. *`srcPath`* must not identify a directory.

`dstPath`

A path that identifies a destination file or directory on the same filesystem as *`srcPath`*.

The destination should not yet exist. The destination path must end in a filename; there is no implicit adoption of the source filename.

`error`

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value

`YES` if the link operation is successful, otherwise `NO`.

Discussion

If pathname *`srcPath`* identifies a file, this method hard-links the file specified in *`dstPath`* to it. If *`srcPath`* is a symbolic link, this method copies it to *`dstPath`* instead of creating a hard link. Symbolic links in *`srcPath`* are not traversed.

Amongst other reasons (such as the disk being full, permissions problems, and so on), this method will fail if:

- *`srcPath`* doesn't point to any file in the file system;
- *`srcPath`* points to an existing symbolic link, but the symbolic link is “broken" (it doesn't in turn point to an existing regular file in the file system);
- *`srcPath`* points to a directory;
- The computer has more than one file system (such as extra partitions, mounted disk images, or network volumes), and *`srcPath`* and *`dstPath`* specify paths in different file systems.

Availability

Available in Mac OS X v10.5 and later.

See Also

`- fileManager:shouldLinkItemAtPath:toPath:` (page 42)
`- fileManager:shouldProceedAfterError:linkingItemAtPath:toPath:` (page 45)
`- createSymbolicLinkAtPath:withDestinationPath:error:` (page 20)
`- copyItemAtPath:toPath:error:` (page 15)
`- moveItemAtPath:toPath:error:` (page 34)
`- linkPath:toPath:handler:` (page 32)

Declared In

`NSFileManager.h`

linkPath:toPath:handler:

Creates a link from a source to a destination.

```
- (BOOL)linkPath:(NSString *)source  toPath:(NSString *)destination
    handler:(id)handler
```

Parameters

source

A path that identifies a source file or directory.

The file, link, or directory specified by *source* must exist.

destination

A path that identifies a destination file or directory.

The destination should not yet exist. The destination path must end in a filename; there is no implicit adoption of the source filename.

handler

An object that responds to the callback messages `fileManager:willProcessPath:` (page 47) and `fileManager:shouldProceedAfterError:` (page 43). You can specify `nil` for *handler*; if you do so and an error occurs, the method automatically returns `NO`.

Return Value

`YES` if the link operation is successful. If the operation is not successful, but the handler method `fileManager:shouldProceedAfterError:` (page 43) returns `YES`, also returns `YES`. Otherwise returns `NO`.

Discussion

If pathname *source* identifies a file, this method hard-links the file specified in *destination* to it. If *source* is a directory or symbolic link, this method copies it to *destination* instead of creating a hard link. Symbolic links in *source* are not traversed.

The handler callback mechanism is similar to delegation. `NSFileManager` sends `fileManager:willProcessPath:` (page 47) when it begins a copy, move, remove, or link operation. It sends `fileManager:shouldProceedAfterError:` (page 43) when it encounters any error in processing

This code fragment verifies the pathname typed in a text field (`documentFileField`) and then links the file to the user's Documents directory:

```
NSString *source = [documentFileField stringValue];

NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
if ([paths count] > 0)
{
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *documentFileName = [source lastPathComponent];
    NSString *destination = [documentsDirectory
stringByAppendingPathComponent:documentFileName];
    NSFileManager *fileManager = [NSFileManager defaultManager];

    if ([fileManager fileExistsAtPath:source])
    {
        [fileManager linkPath:source toPath:destination handler:self];
    }
}
```

Availability

Available in Mac OS X v10.0 and later.

See Also

- `linkItemAtPath:toPath:error:` (page 31)
- `copyPath:toPath:handler:` (page 16)
- `createSymbolicLinkAtPath:pathContent:` (page 20)
- `movePath:toPath:handler:` (page 34)
- `fileManager:shouldProceedAfterError:` (page 43)
- `removeFileAtPath:handler:` (page 36)
- `fileManager:willProcessPath:` (page 47)

Declared In

`NSFileManager.h`

moveItemAtPath:toPath:error:

Moves the directory or file specified by a given path to a different location in the file system identified by another path.

```
- (BOOL)moveItemAtPath:(NSString *)srcPath toPath:(NSString *)dstPath error:(NSError
    **)error
```

Parameters

srcPath
: The path of a file or directory to move. *srcPath* must exist.

dstPath
: The path to which the file or directory at *srcPath* is moved. *destination* must not yet exist. The destination path must end in a filename; there is no implicit adoption of the source filename.

error
: If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value

`YES` if the move operation is successful, otherwise `NO`.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `fileManager:shouldMoveItemAtPath:toPath:` (page 42)
- `fileManager:shouldProceedAfterError:movingItemAtPath:toPath:` (page 45)

Declared In

`NSFileManager.h`

movePath:toPath:handler:

Moves the directory or file specified by a given path to a different location in the file system identified by another path.

```
- (BOOL)movePath:(NSString *)source  toPath:(NSString *)destination
    handler:(id)handler
```

Parameters

`source`

The path of a file or directory to move. *`source`* must exist.

`destination`

The path to which *`source`* is moved. *`destination`* must not yet exist. The destination path must end in a filename; there is no implicit adoption of the source filename.

`handler`

An object that responds to the callback messages `fileManager:willProcessPath:` (page 47) and `fileManager:shouldProceedAfterError:` (page 43). You can specify `nil` for *`handler`*; if you do so and an error occurs, the method automatically returns `NO`.

Return Value

`YES` if the move operation is successful. If the operation is not successful, but the handler method `fileManager:shouldProceedAfterError:` (page 43) returns `YES`, `movePath:toPath:handler:` (page 34) also returns `YES`; otherwise returns `NO`.

Discussion

If *`source`* is a file, the method creates a file at *`destination`* that holds the exact contents of the original file and then deletes the original file. If *`source`* is a directory, `movePath:toPath:handler:` creates a new directory at *`destination`* and recursively populates it with duplicates of the files and directories contained in *`source`*. It then deletes the old directory and its contents. Symbolic links are not traversed, however links are preserved. File or directory attributes—that is, metadata such as owner and group numbers, file permissions, and modification date—are also moved.

The handler callback mechanism is similar to delegation. `NSFileManager` sends `fileManager:willProcessPath:` (page 47) when it begins a copy, move, remove, or link operation. It sends `fileManager:shouldProceedAfterError:` (page 43) when it encounters any error in processing.

If a failure in a move operation occurs, either the preexisting path or the new path remains intact, but not both.

Availability

Available in Mac OS X v10.0 and later.

See Also

– `copyPath:toPath:handler:` (page 16)
– `linkPath:toPath:handler:` (page 32)
– `removeFileAtPath:handler:` (page 36)
– `fileManager:shouldProceedAfterError:` (page 43)
– `fileManager:willProcessPath:` (page 47)

Related Sample Code

QTRecorder

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

WhackedTV

Declared In

`NSFileManager.h`

pathContentOfSymbolicLinkAtPath:

Returns the path of the directory or file that a symbolic link at a given path refers to.

```
- (NSString *)pathContentOfSymbolicLinkAtPath:(NSString *)path
```

Parameters

path
: The path of a symbolic link.

Return Value

The path of the directory or file to which the symbolic link *path* refers, or `nil` upon failure. If the symbolic link is specified as a relative path, that relative path is returned.

Special Considerations

On Mac OS X v10.5 and later, use `destinationOfSymbolicLinkAtPath:error:` (page 22) instead.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- destinationOfSymbolicLinkAtPath:error:` (page 22)
`- createSymbolicLinkAtPath:pathContent:` (page 20)

Declared In

`NSFileManager.h`

removeFileAtPath:handler:

Deletes the file, link, or directory (including, recursively, all subdirectories, files, and links in the directory) identified by a given path.

```
- (BOOL)removeFileAtPath:(NSString *)path  handler:(id)handler
```

Parameters

path
: The path of a file, link, or directory to delete. The value must not be "." or "..".

handler
: An object that responds to the callback messages `fileManager:willProcessPath:` (page 47) and `fileManager:shouldProceedAfterError:` (page 43). You can specify `nil` for *handler*; if you do so and an error occurs, the deletion stops and the method automatically returns `NO`.

Return Value

`YES` if the removal operation is successful. If the operation is not successful, but the handler method `fileManager:shouldProceedAfterError:` (page 43) returns `YES`, also returns `YES`; otherwise returns `NO`.

Discussion

This callback mechanism provided by *handler* is similar to delegation. `NSFileManager` sends `fileManager:willProcessPath:` (page 47) when it begins a copy, move, remove, or link operation. It sends `fileManager:shouldProceedAfterError:` (page 43) when it encounters any error in processing.

Since the removal of directory contents is so thorough and final, be careful when using this method. If you specify "." or ".." for *path* an `NSInvalidArgumentException` exception is raised. This method does not traverse symbolic links.

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

See Also
`– removeItemAtPath:error:` (page 37)
`– copyPath:toPath:handler:` (page 16)
`– linkPath:toPath:handler:` (page 32)
`– movePath:toPath:handler:` (page 34)
`– fileManager:shouldProceedAfterError:` (page 43)
`– fileManager:willProcessPath:` (page 47)

Related Sample Code
AutoUpdater
CIVideoDemoGL
Core Data HTML Store
CustomAtomicStoreSubclass
SampleScannerApp

Declared In
`NSFileManager.h`

removeItemAtPath:error:

Deletes the file, link, or directory (including, recursively, all subdirectories, files, and links in the directory) identified by a given path.

```
- (BOOL)removeItemAtPath:(NSString *)path error:(NSError **)error
```

Parameters

path
The path of a file, link, or directory to delete. The value must not be "." or "..".

error
If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
`YES` if the removal operation is successful, otherwise `NO`.

Discussion
Since the removal of directory contents is so thorough and final, be careful when using this method. If you specify "." or ".." for *path* an `NSInvalidArgumentException` exception is raised. This method does not traverse symbolic links.

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

See Also
`– copyItemAtPath:toPath:error:` (page 15)

- `linkItemAtPath:toPath:error:` (page 31)
- `moveItemAtPath:toPath:error:` (page 34)
- `fileManager:shouldRemoveItemAtPath:` (page 46)
- `fileManager:shouldProceedAfterError:removingItemAtPath:` (page 46)
- `removeFileAtPath:handler:` (page 36)

Related Sample Code
URL CacheInfo

Declared In
`NSFileManager.h`

setAttributes:ofItemAtPath:error:

Sets the attributes of a given file or directory.

```
- (BOOL)setAttributes:(NSDictionary *)attributes ofItemAtPath:(NSString *)path
    error:(NSError **)error
```

Parameters

attributes

A dictionary containing as keys the attributes to set for *path* and as values the corresponding value for the attribute. You can set following: `NSFileBusy`, `NSFileCreationDate`, `NSFileExtensionHidden`, `NSFileGroupOwnerAccountID`, `NSFileGroupOwnerAccountName`, `NSFileHFSCreatorCode`, `NSFileHFSTypeCode`, `NSFileImmutable`, `NSFileModificationDate`, `NSFileOwnerAccountID`, `NSFileOwnerAccountName`, `NSFilePosixPermissions`. You can change single attributes or any combination of attributes; you need not specify keys for all attributes.

path

The path of a file or directory.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
`YES` if *all* changes succeed. If any change fails, returns `NO`, but it is undefined whether any changes actually occurred.

Discussion
As in the POSIX standard, the application either must own the file or directory or must be running as superuser for attribute changes to take effect. The method attempts to make all changes specified in attributes and ignores any rejection of an attempted modification.

The `NSFilePosixPermissions` value must be initialized with the code representing the POSIX file-permissions bit pattern. `NSFileHFSCreatorCode` and `NSFileHFSTypeCode` will only be heeded when *path* specifies a file.

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

Declared In
`NSFileManager.h`

setDelegate:

Sets the delegate for the receiver.

```
- (void)setDelegate:(id)delegate
```

Parameters

delegate
: The delegate for the receiver.

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

Declared In
`NSFileManager.h`

stringWithFileSystemRepresentation:length:

Returns an `NSString` object converted from the C-string representation of a pathname in the current file system.

```
- (NSString *)stringWithFileSystemRepresentation:(const char *)string
    length:(NSUInteger)len
```

Parameters

string
: A C string representation of a pathname.

len
: The number of characters in *string*.

Return Value
An `NSString` object converted from the C-string representation *string* with length *len* of a pathname in the current file system.

Discussion
Use this method if your code receives paths as C strings from system routines.

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

See Also
`- fileSystemRepresentationWithPath:` (page 29)

Declared In
`NSFileManager.h`

subpathsAtPath:

Returns an array that contains (as `NSString` objects) the contents of the directory identified by a given path.

```
- (NSArray *)subpathsAtPath:(NSString *)path
```

Parameters

path

The path of the directory to list.

Return Value

An array that contains (as `NSString` objects) the contents of the directory identified by *path*. If *path* is a symbolic link, `subpathsAtPath:` traverses the link. Returns `nil` if it cannot get the device of the linked-to file.

Discussion

This list of directory contents goes very deep and hence is very useful for large file-system subtrees. The method skips "." and ".."

This method reveals every element of the subtree at *path*, including the contents of file packages (such as applications, nib files, and RTFD files). This code fragment gets the contents of `/System/Library/Fonts` after verifying that the directory exists:

```
BOOL isDir=NO;
NSArray *subpaths;
NSString *fontPath = @"/System/Library/Fonts";
NSFileManager *fileManager = [NSFileManager defaultManager];
if ([fileManager fileExistsAtPath:fontPath isDirectory:&isDir] && isDir)
    subpaths = [fileManager subpathsAtPath:fontPath];
```

Special Considerations

On Mac OS X v10.5 and later, use `subpathsOfDirectoryAtPath:error:` (page 40) instead.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `subpathsOfDirectoryAtPath:error:` (page 40)
- `directoryContentsAtPath:` (page 22)
- `enumeratorAtPath:` (page 24)

Declared In

`NSFileManager.h`

subpathsOfDirectoryAtPath:error:

Returns an array that contains the filenames of the items in the directory specified by a given path and all its subdirectories recursively.

```
- (NSArray *)subpathsOfDirectoryAtPath:(NSString *)path error:(NSError **)error
```

Parameters

path

The path of the directory to list.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
An array that contains `NSString` objects representing the filenames of the items in the directory specified by *`path`* and all its subdirectories recursively. If *`path`* is a symbolic link, `subpathsOfDirectoryAtPath:error:` traverses the link. Returns `nil` if it cannot get the device of the linked-to file.

Discussion
This list of directory contents goes very deep and hence is very useful for large file-system subtrees. The method skips "`.`" and "`..`".

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

See Also
– `subpathsAtPath:` (page 39)
– `directoryContentsAtPath:` (page 22)
– `enumeratorAtPath:` (page 24)

Declared In
`NSFileManager.h`

Delegate Methods

fileManager:shouldCopyItemAtPath:toPath:

An `NSFileManager` object sends this message immediately before attempting to copy to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager shouldCopyItemAtPath:(NSString
    *)srcPath toPath:(NSString *)dstPath
```

Parameters

`fileManager`
    The `NSFileManager` object that sent this message.

`srcPath`
    The path or a file or directory that *`manager`* is about to attempt to copy.

`dstPath`
    The path or a file or directory to which *`manager`* is about to attempt to copy.

Return Value
`YES` if the operation should proceed, otherwise `NO`.

Discussion
You can implement this method in your delegate to monitor file operations.

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

See Also
– `copyItemAtPath:toPath:error:` (page 15)
– `fileManager:shouldProceedAfterError:copyingItemAtPath:toPath:` (page 44)

Declared In
`NSFileManager.h`

fileManager:shouldLinkItemAtPath:toPath:

An `NSFileManager` object sends this message immediately before attempting to link to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager
    shouldLinkItemAtPath:(NSString *)srcPath
    toPath:(NSString *)dstPath
```

Parameters

fileManager
: The `NSFileManager` object that sent this message.

srcPath
: The path or a file or directory that *manager* is about to attempt to link.

dstPath
: The path or a file or directory to which *manager* is about to attempt to link.

Return Value
`YES` if the operation should proceed, otherwise `NO`.

Discussion
You can implement this method in your delegate to monitor file operations.

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

See Also
- `linkItemAtPath:toPath:error:` (page 31)
- `fileManager:shouldProceedAfterError:linkingItemAtPath:toPath:` (page 45)

Declared In
`NSFileManager.h`

fileManager:shouldMoveItemAtPath:toPath:

An `NSFileManager` object sends this message immediately before attempting to move to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager  shouldMoveItemAtPath:(NSString
    *)srcPath  toPath:(NSString *)dstPath
```

Parameters

fileManager
: The `NSFileManager` object that sent this message.

srcPath
: The path or a file or directory that *manager* is about to attempt to move.

dstPath
: The path or a file or directory to which *manager* is about to attempt to move.

Return Value
`YES` if the operation should proceed, otherwise `NO`.

Discussion
You can implement this method in your delegate to monitor file operations.

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

See Also
`- moveItemAtPath:toPath:error:` (page 34)
`- fileManager:shouldProceedAfterError:movingItemAtPath:toPath:` (page 45)

Declared In
`NSFileManager.h`

fileManager:shouldProceedAfterError:

An `NSFileManager` object sends this message to its handler for each error it encounters when copying, moving, removing, or linking files or directories.

```
- (BOOL)fileManager:(NSFileManager *)manager shouldProceedAfterError:(NSDictionary
     *)errorInfo
```

Parameters

`manager`
The file manager that sent this message.

`errorInfo`
A dictionary that contains two or three pieces of information (all `NSString` objects) related to the error:

Key	Value
`@"Path"`	The path related to the error (usually the source path)
`@"Error"`	A description of the error
`@"ToPath"`	The destination path (not all errors)

Return Value
`YES` if the operation (which is often continuous within a loop) should proceed, otherwise `NO`.

Discussion
An `NSFileManager` object, *`manager`*, sends this message for each error it encounters when copying, moving, removing, or linking files or directories. The return value is passed back to the invoker of `copyPath:toPath:handler:` (page 16), `movePath:toPath:handler:` (page 34), `removeFileAtPath:handler:` (page 36), or `linkPath:toPath:handler:` (page 32). If an error occurs and your handler has not implemented this method, the invoking method automatically returns `NO`.

The following implementation of `fileManager:shouldProceedAfterError:` displays the error string in an alert dialog and leaves it to the user whether to proceed or stop:

```
-(BOOL)fileManager:(NSFileManager *)manager
        shouldProceedAfterError:(NSDictionary *)errorInfo
{
    int result;
    result = NSRunAlertPanel(@"Gumby App", @"File  operation error:
            %@ with file: %@", @"Proceed", @"Stop",  NULL,
            [errorInfo objectForKey:@"Error"],
            [errorInfo objectForKey:@"Path"]);

    if (result == NSAlertDefaultReturn)
        return YES;
    else
        return NO;
}
```

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

See Also
- `fileManager:willProcessPath:` (page 47)

Declared In
`NSFileManager.h`

fileManager:shouldProceedAfterError:copyingItemAtPath:toPath:

An `NSFileManager` object sends this message if an error occurs during an attempt to copy to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager shouldProceedAfterError:(NSError
    *)error copyingItemAtPath:(NSString *)srcPath toPath:(NSString *)dstPath
```

Parameters

fileManager
: The `NSFileManager` object that sent this message.

error
: The error that occurred during the attempt to copy.

srcPath
: The path or a file or directory that *manager* is attempting to copy.

dstPath
: The path or a file or directory to which *manager* is attempting to copy.

Return Value
`YES` if the operation should proceed, otherwise `NO`.

Discussion
You can implement this method in your delegate to monitor file operations.

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

See Also
- `copyItemAtPath:toPath:error:` (page 15)
- `fileManager:shouldCopyItemAtPath:toPath:` (page 41)

Declared In
`NSFileManager.h`

fileManager:shouldProceedAfterError:linkingItemAtPath:toPath:

An `NSFileManager` object sends this message if an error occurs during an attempt to link to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager
    shouldProceedAfterError:(NSError *)error
    linkingItemAtPath:(NSString *)srcPath
    toPath:(NSString *)dstPath
```

Parameters

fileManager

The `NSFileManager` object that sent this message.

error

The error that occurred during the attempt to link.

srcPath

The path or a file or directory that *manager* is attempting to link.

dstPath

The path or a file or directory to which *manager* is attempting to link.

Return Value

`YES` if the operation should proceed, otherwise `NO`.

Availability

Available in Mac OS X v10.5 and later.

See Also

`– linkItemAtPath:toPath:error:` (page 31)
`– fileManager:shouldLinkItemAtPath:toPath:` (page 42)

Declared In

`NSFileManager.h`

fileManager:shouldProceedAfterError:movingItemAtPath:toPath:

An `NSFileManager` object sends this message if an error occurs during an attempt to move to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager
    shouldProceedAfterError:(NSError *)error
    movingItemAtPath:(NSString *)srcPath
    toPath:(NSString *)dstPath
```

Parameters

fileManager

The `NSFileManager` object that sent this message.

error

The error that occurred during the attempt to move.

srcPath

The path or a file or directory that *manager* is attempting to move.

dstPath

The path or a file or directory to which *manager* is attempting to move.

Return Value
YES if the operation should proceed, otherwise NO.

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

See Also
- moveItemAtPath:toPath:error: (page 34)
- fileManager:shouldMoveItemAtPath:toPath: (page 42)

Declared In
NSFileManager.h

fileManager:shouldProceedAfterError:removingItemAtPath:

An NSFileManager object sends this message if an error occurs during an attempt to delete a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager
    shouldProceedAfterError:(NSError *)error
    removingItemAtPath:(NSString *)path
```

Parameters

fileManager
The NSFileManager object that sent this message.

error
The error that occurred during the attempt to copy.

path
The path or a file or directory that *manager* is attempting to delete.

Return Value
YES if the operation should proceed, otherwise NO.

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

See Also
- removeItemAtPath:error: (page 37)
- fileManager:shouldRemoveItemAtPath: (page 46)

Declared In
NSFileManager.h

fileManager:shouldRemoveItemAtPath:

An NSFileManager object sends this message immediately before attempting to delete an item at a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager
    shouldRemoveItemAtPath:(NSString *)path
```

Parameters

`fileManager`
: The `NSFileManager` object that sent this message.

`path`
: The path or a file or directory that *`manager`* is about to attempt to delete.

Return Value

`YES` if the operation should proceed, otherwise `NO`.

Discussion

You can implement this method in your delegate to monitor file operations.

Availability

Available in Mac OS X v10.5 and later.

See Also

`- removeItemAtPath:error:` (page 37)
`- fileManager:shouldProceedAfterError:removingItemAtPath:` (page 46)

Declared In

`NSFileManager.h`

fileManager:willProcessPath:

An `NSFileManager` object sends this message to a handler immediately before attempting to move, copy, rename, or delete, or before attempting to link to a given path.

```
- (void)fileManager:(NSFileManager *)manager  willProcessPath:(NSString *)path
```

Parameters

`manager`
: The `NSFileManager` object that sent this message.

`path`
: The path or a file or directory that *`manager`* is about to attempt to move, copy, rename, delete, or link to.

Discussion

You can implement this method in your handler to monitor file operations.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSFileManager.h`

Constants

File Attribute Keys

These keys access file attribute values contained in `NSDictionary` objects used by `changeFileAttributes:atPath:` (page 12), `fileAttributesAtPath:traverseLink:` (page 25), `createDirectoryAtPath:attributes:` (page 17), and `createFileAtPath:contents:attributes:` (page 19).

```
NSString *NSFileType;
NSString *NSFileTypeDirectory;
NSString *NSFileTypeRegular;
NSString *NSFileTypeSymbolicLink;
NSString *NSFileTypeSocket;
NSString *NSFileTypeCharacterSpecial;
NSString *NSFileTypeBlockSpecial;
NSString *NSFileTypeUnknown;
NSString *NSFileSize;
NSString *NSFileModificationDate;
NSString *NSFileReferenceCount;
NSString *NSFileDeviceIdentifier;
NSString *NSFileOwnerAccountName;
NSString *NSFileGroupOwnerAccountName;
NSString *NSFilePosixPermissions;
NSString *NSFileSystemNumber;
NSString *NSFileSystemFileNumber;
NSString *NSFileExtensionHidden;
NSString *NSFileHFSCreatorCode;
NSString *NSFileHFSTypeCode;
NSString *NSFileImmutable;
NSString *NSFileAppendOnly;
NSString *NSFileCreationDate;
NSString *NSFileOwnerAccountID;
NSString *NSFileGroupOwnerAccountID;
NSString *NSFileBusy;
```

Constants

`NSFileAppendOnly`

The key in a file attribute dictionary whose value indicates whether the file is read-only.

The corresponding value is an `NSNumber` object containing a Boolean value.

Available in Mac OS X v10.2 and later.

Declared in `NSFileManager.h`.

`NSFileBusy`

The key in a file attribute dictionary whose value indicates whether the file is busy.

The corresponding value is an `NSNumber` object containing a Boolean value.

Available in Mac OS X v10.4 and later.

Declared in `NSFileManager.h`.

`NSFileCreationDate`
: The key in a file attribute dictionary whose value indicates the file's creation date.

  The corresponding value is an `NSDate` object.

  Available in Mac OS X v10.2 and later.

  Declared in `NSFileManager.h`.

`NSFileOwnerAccountName`
: The key in a file attribute dictionary whose value indicates the name of the file's owner.

  The corresponding value is an `NSString` object.

  Available in Mac OS X v10.0 and later.

  Declared in `NSFileManager.h`.

`NSFileGroupOwnerAccountName`
: The key in a file attribute dictionary whose value indicates the group name of the file's owner.

  The corresponding value is an `NSString` object.

  Available in Mac OS X v10.0 and later.

  Declared in `NSFileManager.h`.

`NSFileDeviceIdentifier`
: The key in a file attribute dictionary whose value indicates the identifier for the device on which the file resides.

  The corresponding value is an `NSNumber` object containing an `unsigned long`.

  Available in Mac OS X v10.0 and later.

  Declared in `NSFileManager.h`.

`NSFileExtensionHidden`
: The key in a file attribute dictionary whose value indicates whether the file's extension is hidden.

  The corresponding value is an `NSNumber` object containing a Boolean value.

  Available in Mac OS X v10.1 and later.

  Declared in `NSFileManager.h`.

`NSFileGroupOwnerAccountID`
: The key in a file attribute dictionary whose value indicates the file's group ID.

  The corresponding value is an `NSNumber` object containing an `unsigned long`.

  Available in Mac OS X v10.2 and later.

  Declared in `NSFileManager.h`.

`NSFileHFSCreatorCode`
: The key in a file attribute dictionary whose value indicates the file's HFS creator code.

  The corresponding value is an `NSNumber` object containing an `unsigned long`. See HFS File Types for possible values.

  Available in Mac OS X v10.1 and later.

  Declared in `NSFileManager.h`.

`NSFileHFSTypeCode`
: The key in a file attribute dictionary whose value indicates the file's HFS type code.

  The corresponding value is an `NSNumber` object containing an `unsigned long`. See HFS File Types for possible values.

  Available in Mac OS X v10.1 and later.

  Declared in `NSFileManager.h`.

`NSFileImmutable`
The key in a file attribute dictionary whose value indicates whether the file is mutable.

The corresponding value is an `NSNumber` object containing a Boolean value.

Available in Mac OS X v10.2 and later.

Declared in `NSFileManager.h`.

`NSFileModificationDate`
The key in a file attribute dictionary whose value indicates the file's last modified date.

The corresponding value is an `NSDate` object.

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileOwnerAccountID`
The key in a file attribute dictionary whose value indicates the file's owner's account ID.

The corresponding value is an `NSNumber` object containing an `unsigned long`.

Available in Mac OS X v10.2 and later.

Declared in `NSFileManager.h`.

`NSFilePosixPermissions`
The key in a file attribute dictionary whose value indicates the file's Posix permissions.

The corresponding value is an `NSNumber` object containing an `unsigned long`.

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileReferenceCount`
The key in a file attribute dictionary whose value indicates the file's reference count.

The corresponding value is an `NSNumber` object containing an `unsigned long`.

The number specifies the number of hard links to a file.

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileSize`
The key in a file attribute dictionary whose value indicates the file's size in bytes.

The corresponding value is an `NSNumber` object containing an `unsigned long long`.

Important: If the file has a resource fork, the returned value does *not* include the size of the resource fork.

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileSystemFileNumber`
The key in a file attribute dictionary whose value indicates the file's filesystem file number.

The corresponding value is an `NSNumber` object containing an `unsigned long`. The value corresponds to the value of `st_ino`, as returned by `stat`(2).

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileType`
The key in a file attribute dictionary whose value indicates the file's type.

The corresponding value is an `NSString` object (see below for possible values).

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

Discussion
`NSFileDeviceIdentifier` is used to access the identifier of a remote device.

Declared In
`NSFileManager.h`

File Type Attribute Keys

These strings the possible values for the `NSFileType` attribute key contained in the `NSDictionary` object returned from `NSFileManager`'s `fileAttributesAtPath:traverseLink:` (page 25).

```
extern NSString *NSFileTypeDirectory;
extern NSString *NSFileTypeRegular;
extern NSString *NSFileTypeSymbolicLink;
extern NSString *NSFileTypeSocket;
extern NSString *NSFileTypeCharacterSpecial;
extern NSString *NSFileTypeBlockSpecial;
extern NSString *NSFileTypeUnknown;
```

Constants
`NSFileTypeDirectory`
Directory

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileTypeRegular`
Regular file

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileTypeSymbolicLink`
Symbolic link

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileTypeSocket`
Socket

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileTypeCharacterSpecial`
Character special file

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileTypeBlockSpecial`

Block special file

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileTypeUnknown`

Unknown

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

Declared In

`NSFileManager.h`

File-System Attribute Keys

Keys to access the file attribute values contained in the `NSDictionary` object returned from `NSFileManager`'s `fileSystemAttributesAtPath:` (page 28) method.

```
extern NSString *NSFileSystemSize;
extern NSString *NSFileSystemFreeSize;
extern NSString *NSFileSystemNodes;
extern NSString *NSFileSystemFreeNodes;
extern NSString *NSFileSystemNumber;
```

Constants

`NSFileSystemSize`

The key in a file system attribute dictionary whose value indicates the size of the file system.

The corresponding value is an `NSNumber` object that specifies the size of the file system in bytes. The value is determined by `statfs()`.

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileSystemFreeSize`

The key in a file system attribute dictionary whose value indicates the amount of free space on the file system.

The corresponding value is an `NSNumber` object that specifies the amount of free space on the file system in bytes. The value is determined by `statfs()`.

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileSystemNodes`

The key in a file system attribute dictionary whose value indicates the number of nodes in the file system.

The corresponding value is an `NSNumber` object that specifies the number of nodes in the file system.

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileSystemFreeNodes`

The key in a file system attribute dictionary dictionary whose value indicates the number of free nodes in the file system.

The corresponding value is an `NSNumber` object that specifies the number of free nodes in the file system.

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileSystemNumber`

The key in a file system attribute dictionary dictionary whose value indicates the filesystem number of the file system.

The corresponding value is an `NSNumber` object that specifies the filesystem number of the file system. The value corresponds to the value of `st_dev`, as returned by `stat`(2).

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

Declared In
`NSFileManager.h`

Resource Fork Support

Specifies the version of the Foundation framework in which `NSFileManager` first supported resource forks.

```
#define NSFoundationVersionWithFileManagerResourceForkSupport 412
```

Constants

`NSFoundationVersionWithFileManagerResourceForkSupport`

The version of the Foundation framework in which `NSFileManager` first supported resource forks.

Available in Mac OS X v10.1 and later.

Declared in `NSFileManager.h`.

Declared In
`NSFileManager.h`

Document Revision History

This table describes the changes to *NSFileManager Class Reference*.

Date	Notes
2008-10-15	Corrected typographical errors.
2007-12-11	Corrected minor errors.
2007-10-31	Made several minor corrections.
2007-03-12	Updated to include API introduced in Mac OS X v10.5.
2006-06-28	Corrected typographical errors.
2006-05-23	Corrected declarations for NSFileOwnerAccountName and NSFileGroupOwnerAccountName.
	First publication of this content as a separate document.

Index