Class

FileManager

A convenient interface to the contents of the file system, and the primary means of interacting with it.

iOS 2.0+ | iPadOS 2.0+ | Mac Catalyst 13.0+ | macOS 10.0+ | tvOS 9.0+ | visionOS 1.0+ | watchOS 2.0+

```
class FileManager
```

Mentioned in

- About Apple File System
- Using the file system effectively
- Optimizing Your App's Data for iCloud Backup

Overview

A file manager object lets you examine the contents of the file system and make changes to it. The `FileManager` class provides convenient access to a shared file manager object that is suitable for most types of file-related manipulations. A file manager object is typically your primary mode of interaction with the file system. You use it to locate, create, copy, and move files and directories. You also use it to get information about a file or directory or change some of its attributes.

When specifying the location of files, you can use either NSURL or `NSString` objects. The use of the NSURL class is generally preferred for specifying file-system items because URLs can convert path information to a more efficient representation internally. You can also obtain a bookmark from an NSURL object, which is similar to an alias and offers a more sure way of locating the file or directory later.

If you are moving, copying, linking, or removing files or directories, you can use a delegate in conjunction with a file manager object to manage those operations. The delegate's role is to affirm the operation and to decide whether to proceed when errors occur. In macOS 10.7 and later, the delegate must conform to the `FileManagerDelegate` protocol.

In iOS 5.0 and later and in macOS 10.7 and later, `FileManager` includes methods for managing items stored in iCloud. Files and directories tagged for cloud storage are synced to iCloud so that they can be made available to the user's iOS devices and Macintosh computers. Changes to an item in one location are propagated to all other locations to ensure the items stay in sync.

Sync control

A package is a directory that the system presents as a single file to the person using the device. Apps with documents that contain multiple files can use packages to manage contents like media assets. In iOS 26 and macOS 26 and later, `FileManager` introduces methods for controlling how a file provider syncs these items. By pausing sync when your app opens a package and resuming when it closes, your app can prevent the file provider from changing the contents of the package in unexpected ways, which potentially leaves the document in an inconsistent state. You can also use this pause and resume API on regular "flat" files.

Threading considerations

The methods of the shared `FileManager` object can be called from multiple threads safely. However, if you use a delegate to receive notifications about the status of move, copy, remove, and link operations, you should create a unique instance of the file manager object, assign your delegate to that object, and use that file manager to initiate your operations.

Topics

Creating a file manager

```
convenience init(authorization: NSWorkspace.Authorization)
```

Initializes a file manager object that is authorized to perform privileged file system operations.

```
class var `default`: FileManager
```

The shared file manager object for the process.

Accessing user directories

```
var homeDirectoryForCurrentUser: URL
```

The home directory for the current user.

```
func NSHomeDirectory() -> String
```

Returns the path to either the user's or application's home directory, depending on the platform.

```
func NSUserName() -> String
```

Returns the logon name of the current user.

```
func NSFullUserName() -> String
```

Returns a string containing the full name of the current user.

```
func homeDirectory(forUser: String) -> URL?
```

Returns the home directory for the specified user.

```
func NSHomeDirectoryForUser(String?) -> String?
```

Returns the path to a given user's home directory.

```
var temporaryDirectory: URL
```

The temporary directory for the current user.

```
func NSTemporaryDirectory() -> String
```

Returns the path of the temporary directory for the current user.

Locating system directories

```
func url(for: FileManager.SearchPathDirectory, in: FileManager.Search
PathDomainMask, appropriateFor: URL?, create: Bool) throws -> URL
```

Locates and optionally creates the specified common directory in a domain.

```
func urls(for: FileManager.SearchPathDirectory, in: FileManager.Search
PathDomainMask) -> [URL]
```

Returns an array of URLs for the specified common directory in the requested domains.

```
func NSSearchPathForDirectoriesInDomains(FileManager.SearchPath
Directory, FileManager.SearchPathDomainMask, Bool) -> [String]
```

Creates a list of directory search paths.

```
func NSOpenStepRootDirectory() -> String
```

Returns the root directory of the user's system.

Locating application group container directories

`func containerURL(forSecurityApplicationGroupIdentifier: String) -> URL?`

Returns the container directory associated with the specified security application group identifier.

`App Groups Entitlement`

A list of identifiers specifying the groups your app belongs to.

Discovering directory contents

`func contentsOfDirectory(at: URL, includingPropertiesForKeys: [URLResourceKey]?, options: FileManager.DirectoryEnumerationOptions) throws -> [URL]`

Performs a shallow search of the specified directory and returns URLs for the contained items.

`func contentsOfDirectory(atPath: String) throws -> [String]`

Performs a shallow search of the specified directory and returns the paths of any contained items.

`func enumerator(at: URL, includingPropertiesForKeys: [URLResourceKey]?, options: FileManager.DirectoryEnumerationOptions, errorHandler: ((URL, any Error) -> Bool)?) -> FileManager.DirectoryEnumerator?`

Returns a directory enumerator object that can be used to perform a deep enumeration of the directory at the specified URL.

`func enumerator(atPath: String) -> FileManager.DirectoryEnumerator?`

Returns a directory enumerator object that can be used to perform a deep enumeration of the directory at the specified path.

`class DirectoryEnumerator`

An object that enumerates the contents of a directory.

`func mountedVolumeURLs(includingResourceValuesForKeys: [URLResourceKey]?, options: FileManager.VolumeEnumerationOptions) -> [URL]?`

Returns an array of URLs that identify the mounted volumes available on the device.

`struct VolumeEnumerationOptions`

Options for enumerating mounted volumes with the `mountedVolumeURLs(including ResourceValuesForKeys:options:)` method.

```
func subpathsOfDirectory(atPath: String) throws -> [String]
```

Performs a deep enumeration of the specified directory and returns the paths of all of the contained subdirectories.

```
func subpaths(atPath: String) -> [String]?
```

Returns an array of strings identifying the paths for all items in the specified directory.

Creating and deleting items

```
func createDirectory(at: URL, withIntermediateDirectories: Bool, attributes: [FileAttributeKey : Any]?) throws
```

Creates a directory with the given attributes at the specified URL.

```
func createDirectory(atPath: String, withIntermediateDirectories: Bool, attributes: [FileAttributeKey : Any]?) throws
```

Creates a directory with given attributes at the specified path.

```
func createFile(atPath: String, contents: Data?, attributes: [FileAttributeKey : Any]?) -> Bool
```

Creates a file with the specified content and attributes at the given location.

```
func removeItem(at: URL) throws
```

Removes the file or directory at the specified URL.

```
func removeItem(atPath: String) throws
```

Removes the file or directory at the specified path.

```
func trashItem(at: URL, resultingItemURL: AutoreleasingUnsafeMutablePointer<NSURL?>?) throws
```

Moves an item to the trash.

Replacing items

```
func replaceItemAt(URL, withItemAt: URL, backupItemName: String?, options: FileManager.ItemReplacementOptions) throws -> URL?
```

Replaces the contents of the item at the specified URL in a manner that ensures no data loss occurs.

`func replaceItem(at: URL, withItemAt: URL, backupItemName: String?, options: FileManager.ItemReplacementOptions, resultingItemURL: AutoreleasingUnsafeMutablePointer<NSURL?>?) throws`

Replaces the contents of the item at the specified URL in a manner that ensures no data loss occurs.

`struct ItemReplacementOptions`

Options for specifying the behavior of file replacement operations.

Moving and copying items

`func copyItem(at: URL, to: URL) throws`

Copies the file at the specified URL to a new location synchronously.

`func copyItem(atPath: String, toPath: String) throws`

Copies the item at the specified path to a new location synchronously.

`func moveItem(at: URL, to: URL) throws`

Moves the file or directory at the specified URL to a new location synchronously.

`func moveItem(atPath: String, toPath: String) throws`

Moves the file or directory at the specified path to a new location synchronously.

Managing iCloud-based items

`var ubiquityIdentityToken: (any NSCoding & NSCopying & NSObjectProtocol)?`

An opaque token that represents the current user's iCloud Drive Documents identity.

`func url(forUbiquityContainerIdentifier: String?) -> URL?`

Returns the URL for the iCloud container associated with the specified identifier and establishes access to that container.

`func isUbiquitousItem(at: URL) -> Bool`

Returns a Boolean indicating whether the item is targeted for storage in iCloud.

`func setUbiquitous(Bool, itemAt: URL, destinationURL: URL) throws`

Indicates whether the item at the specified URL should be stored in iCloud.

`func startDownloadingUbiquitousItem(at: URL) throws`

Starts downloading (if necessary) the specified item to the local system.

`func evictUbiquitousItem(at: URL) throws`

Removes the local copy of the specified item that's stored in iCloud.

`func url(forPublishingUbiquitousItemAt: URL, expiration: Autoreleasing UnsafeMutablePointer<NSDate?>?) throws -> URL`

Returns a URL that can be emailed to users to allow them to download a copy of a flat file item from iCloud.

Accessing file provider services

`func getFileProviderServicesForItem(at: URL, completionHandler: ([NSFileProviderServiceName : NSFileProviderService]?, (any Error)?) -> Void)`

Returns the services provided by the File Provider extension that manages the item at the given URL.

`class NSFileProviderService`

A service that provides a custom communication channel between your app and a File Provider extension.

`struct NSFileProviderServiceName`

The name used to identify a File Provider service.

Controlling file provider synchronization

`struct NSFileManagerSupportedSyncControls`

An option set of the sync controls available for an item.

`func pauseSyncForUbiquitousItem(at: URL, completionHandler: ((any Error)?) -> Void)`

Asynchronously pauses sync of an item at the given URL.

`func resumeSyncForUbiquitousItem(at: URL, with: NSFileManagerResumeSyncBehavior, completionHandler: ((any Error)?) -> Void)`

Asynchronously resumes the sync on a paused item using the given resume behavior.

`enum NSFileManagerResumeSyncBehavior`

The behaviors the file manager can apply to resolve conflicts when resuming a sync.

`func fetchLatestRemoteVersionOfItem(at: URL, completionHandler: (NSFileVersion?, (any Error)?) -> Void)`

Asynchronously fetches the latest remote version of a given item from the server.

```
class NSFileVersion
```

A snapshot of a file at a specific point in time.

```
func uploadLocalVersionOfUbiquitousItem(at: URL, withConflictResolutionPolicy: NSFileManagerUploadLocalVersionConflictPolicy, completionHandler: (NSFileVersion?, (any Error)?) -> Void)
```

Asynchronously uploads the local version of the item using the provided conflict resolution policy.

```
enum NSFileManagerUploadLocalVersionConflictPolicy
```

The policies the file manager can apply to resolve conflicts when uploading a local version of a file.

Creating symbolic and hard links

```
func createSymbolicLink(at: URL, withDestinationURL: URL) throws
```

Creates a symbolic link at the specified URL that points to an item at the given URL.

```
func createSymbolicLink(atPath: String, withDestinationPath: String) throws
```

Creates a symbolic link that points to the specified destination.

```
func linkItem(at: URL, to: URL) throws
```

Creates a hard link between the items at the specified URLs.

```
func linkItem(atPath: String, toPath: String) throws
```

Creates a hard link between the items at the specified paths.

```
func destinationOfSymbolicLink(atPath: String) throws -> String
```

Returns the path of the item pointed to by a symbolic link.

Determining access to files

```
func fileExists(atPath: String) -> Bool
```

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

```
func fileExists(atPath: String, isDirectory: UnsafeMutablePointer<ObjCBool>?) -> Bool
```

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

```
func isReadableFile(atPath: String) -> Bool
```

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

```
func isWritableFile(atPath: String) -> Bool
```

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

```
func isExecutableFile(atPath: String) -> Bool
```

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

```
func isDeletableFile(atPath: String) -> Bool
```

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

Getting and setting attributes

```
func componentsToDisplay(forPath: String) -> [String]?
```

Returns an array of strings representing the user-visible components of a given path.

```
func displayName(atPath: String) -> String
```

Returns the display name of the file or directory at a specified path.

```
func attributesOfItem(atPath: String) throws -> [FileAttributeKey : Any
]
```

Returns the attributes of the item at a given path.

```
func attributesOfFileSystem(forPath: String) throws -> [FileAttribute
Key : Any]
```

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

```
func setAttributes([FileAttributeKey : Any], ofItemAtPath: String)
throws
```

Sets the attributes of the specified file or directory.

Getting and comparing file contents

```
func contents(atPath: String) -> Data?
```

Returns the contents of the file at the specified path.

```
func contentsEqual(atPath: String, andPath: String) -> Bool
```

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

Getting the relationship between items

```
func getRelationship(UnsafeMutablePointer<FileManager.URLRelationship>, ofDirectoryAt: URL, toItemAt: URL) throws
```

Determines the type of relationship that exists between a directory and an item.

```
func getRelationship(UnsafeMutablePointer<FileManager.URLRelationship>, of: FileManager.SearchPathDirectory, in: FileManager.SearchPathDomainMask, toItemAt: URL) throws
```

Determines the type of relationship that exists between a system directory and the specified item.

```
enum URLRelationship
```

Constants indicating the relationship between a directory and an item.

Converting file paths to strings

```
func fileSystemRepresentation(withPath: String) -> UnsafePointer<CChar>
```

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

```
func string(withFileSystemRepresentation: UnsafePointer<CChar>, length: Int) -> String
```

Returns an `NSString` object whose contents are derived from the specified C-string path.

Managing the delegate

```
var delegate: (any FileManagerDelegate)?
```

The delegate of the file manager object.

Managing the current directory

```
func changeCurrentDirectoryPath(String) -> Bool
```

Changes the path of the current working directory to the specified path.

```
var currentDirectoryPath: String
```

The path to the program's current directory.

Unmounting volumes

```
func unmountVolume(at: URL, options: FileManager.UnmountOptions,
completionHandler: ((any Error)?) -> Void)
```

Starts the process of unmounting the specified volume.

```
struct UnmountOptions
```

Options that specify the behavior of an unmount operation.

```
let NSFileManagerUnmountDissentingProcessIdentifierErrorKey: String
```

The process identifier of the process that prevented a volume from unmounting.

Working with HFS file types

```
func NSFileTypeForHFSTypeCode(OSType) -> String!
```

Returns a string encoding a file type code.

```
func NSHFSTypeCodeFromFileType(String!) -> OSType
```

Returns a file type code.

```
func NSHFSTypeOfFile(String!) -> String!
```

Returns a string encoding a file type.

Determining resource fork support

```
var NSFoundationVersionWithFileManagerResourceForkSupport: Int32
```

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

Supporting Types

```
struct DirectoryEnumerationOptions
```

Options for enumerating the contents of directories.

```
enum SearchPathDirectory
```

The location of significant directories.

```
struct SearchPathDomainMask
```

Domain constants specifying base locations to use when you search for significant directories.

`struct FileAttributeKey`

Keys in dictionaries used to get and set file attributes.

`struct FileAttributeType`

Values representing a file's type attribute.

`struct FileProtectionType`

Protection level values that can be associated with a file attribute key.

`struct URLFileProtection`

Protection-level values for a URL resource key.

Notifications

`static let NSUbiquityIdentityDidChange: NSNotification.Name`

Sent after the iCloud ("ubiquity") identity has changed.

Deprecated Methods

~~`func changeFileAttributes([AnyHashable : Any], atPath: String) -> Bool`~~

Changes the attributes of a given file or directory.

Deprecated

~~`func fileAttributes(atPath: String, traverseLink: Bool) -> [AnyHashable : Any]?`~~

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

Deprecated

~~`func fileSystemAttributes(atPath: String) -> [AnyHashable : Any]?`~~

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

Deprecated

~~`func directoryContents(atPath: String) -> [Any]?`~~

Returns the directories and files (including symbolic links) contained in a given directory.

Deprecated

~~`func createDirectory(atPath: String, attributes: [AnyHashable : Any]) -> Bool`~~

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

Deprecated

~~`func createSymbolicLink(atPath: String, pathContent: String) -> Bool`~~

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

Deprecated

~~`func pathContentOfSymbolicLink(atPath: String) -> String?`~~

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

Deprecated

~~`func fileManager(_ fm: FileManager, shouldProceedAfterError errorInfo: [AnyHashable : Any]) -> Bool`~~

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

Deprecated

~~`func fileManager(_ fm: FileManager, willProcessPath path: String)`~~

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.

Deprecated

~~`func replaceItemAtURL(originalItemURL: NSURL, withItemAtURL: NSURL, backupItemName: String?, options: FileManager.ItemReplacementOptions) throws -> NSURL?`~~

Replaces the contents of the item at the specified URL in a manner that ensures no data loss occurs.

Deprecated

Structures

`struct UbiquityIdentityDidChangeMessage`

Instance Methods

`func replaceItemAt(URL, withItemAt: URL, backupItemName: String?, options: FileManager.ItemReplacementOptions) throws -> NSURL?`

Relationships

Inherits From

`NSObject`

Conforms To

`CVarArg`
`CustomDebugStringConvertible`
`CustomStringConvertible`
`Equatable`
`Hashable`
`NSObjectProtocol`

See Also

File system operations

- Improving performance and stability when accessing the file system

  Prevent data loss and app crashes by interacting with the file system in a coordinated, asynchronous manner and by avoiding unnecessary disk I/O.

- Using the file system effectively

  Gain access to benefits like automatic backup or purging by using purpose-built directories provided by the system.

`protocol FileManagerDelegate`

The interface a file manager's delegate uses to intervene during operations or if an error occurs.

- About Apple File System

  Use high-level APIs to get the most out of Apple File System.