NSFileWrapper

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	AppKit/NSFileWrapper.h

Class Description

An NSFileWrapper holds a file's contents in dynamic memory. In this role it enables a document object to embed a file, treating it as a unit of data that can be displayed as an image (and possibly edited in place), saved to disk, or transmitted to another application. It can also store an icon for representing the file in a document or in a dragging operation.

Instances of this class are referred to as *file wrapper objects*, and when no confusion will result, merely as *file wrappers*. A file wrapper can be one of three specific types: a *regular file wrapper*, which holds the contents of a single actual file; a *directory wrapper*, which holds a directory and all of the files or directories within it; or a *link wrapper*, which simply represents a symbolic link in the file system (sometimes called a shortcut or alias). Some NSFileWrapper methods apply only to a specific type, and raise an exception if sent to a file wrapper of the wrong type. To determine the type of a file wrapper, use the **isRegularFile**, **isDirectory**, and **isSymbolicLink** methods.

You can create a file wrapper from data in memory using **initWithSerializedRepresentation:** or from data on disk using **initWithPath:**. Both create the appropriate type of file wrapper based on the nature of the serialized representation or of the file on disk. Three convenience methods each create a file wrapper of a specific type: **initRegularFileWithContents:**, **initDirectoryWithFileWrappers:**, and **initSymbolicLinkWithDestination:**. Because each initialization method creates file wrappers of different types or states, they're all designated initializers for this class—subclasses must meaningfully override them all as necessary.

Memory and Disk Representations

Because the purpose of a file wrapper is to represent files in memory, it's very loosely coupled to any disk representation. A file wrapper doesn't record the path to the disk representation of its contents. This allows you to save the same file wrapper with different paths, but it also requires you to record those paths if you want to update the file wrapper from disk later. NSFileWrapper allows you to set a preferred filename for save operations and records the last filename it was actually saved to; the **preferredFilename** and **filename** methods return these names. This feature is more important for directory wrappers, though, and so is discussed below under "Working with Directory Wrappers."

A file wrapper stores file system information (such as modification time and access permissions), which it updates when reading from disk and uses when writing files to disk. The **fileAttributes** method returns this information in the format described in the NSFileManager class specification. You can also set the file attributes using the **setFileAttributes:** method.

When saving a file wrapper to disk, you typically determine the directory you want to save it in, then append the preferred filename to that directory path and use **writeToFile:atomically:updateFilenames:**, which saves the file wrapper's contents and updates the file attributes. You can save a file wrapper under a different name if you wish, but this may result in the recorded filename differing from the preferred filename, depending on how you invoke the **writeToFile:...** method.

Besides saving its contents to disk, a file wrapper can re-read them from disk when necessary. The **needsToBeUpdatedFromPath:** method determines whether a disk representation may have changed, based on the file attributes stored the last time the file was read or written. If the file wrapper's modification time or access permissions are different from those of the file on disk, this method returns YES. You can then use **updateFromPath:** to re-read the file from disk.

Finally, to transmit a file wrapper to another process or system (for example, over a distributed objects connection or through the pasteboard), you use the **serializedRepresentation** method to get an NSData object containing the file wrapper's contents in the NSFileContentsPboardType format. You can safely transmit this representation over whatever channel you desire. The recipient of the representation can then reconstitute the file wrapper using the **initWithSerializedRepresentation:** method.

Working with Directory Wrappers

A directory wrapper contains other file wrappers (of any type), and allows you to access them by keys derived from their preferred filenames. You can add any type of file wrapper to a directory wrapper with **addFileWrapper:** or **addFileWithPath:**, and remove it with **removeFileWrapper:**. The convenience methods **addRegularFileWithContents:preferredFilename:** and **addSymbolicLinkWithDestination:preferredFilename:** allow you to add regular file and link wrappers while also setting their preferred names.

A directory wrapper stores its contents in an NSDictionary, which you can retrieve using the **fileWrappers** method. The keys of this dictionary are based on the preferred filenames of each file wrapper contained in the directory wrapper. There exist, then, three identifiers for a file wrapper within a directory wrapper:

- Preferred filename. This doesn't uniquely identify the file wrapper, but the following identifiers are always based on it.
- Dictionary key. This is always equal to the preferred name when there are no other file wrappers of the same preferred name in the same directory wrapper. Otherwise, it's a string made by adding a unique prefix to the preferred filename (note that the same file wrapper can have a different dictionary key for each directory wrapper that contains it). You use the dictionary key to retrieve the file wrapper object in memory, in order to get its contents or its filename (to update it from disk). You can get a file wrapper's dictionary key by sending a **keyForFileWrapper:** message to the directory wrapper that contains it.

- Filename. This is usually based on the preferred filename, but isn't necessarily the same as it or the dictionary key. You use the filename to update a single file wrapper relative to the path of the directory wrapper that contains it. Note that the filename may change whenever you save a directory wrapper containing the file wrapper (particularly if the file wrapper has been added to several different directory wrappers); thus, you should always retrieve the filename from the file wrapper itself each time you need it rather than caching it.

When working with the contents of a directory wrapper, you can use a dictionary enumerator to retrieve each file wrapper and perform whatever operation you need. Note that with the exceptions of saving and updating, a directory file wrapper defines no recursive operations for its contents. To set the file attributes for all contained file wrappers, or to perform any other such operation, you must define a recursive method that examines the type of each file wrapper and invokes itself anew for any directory wrapper it encounters.

Method Types

Initializing a file wrapper	– initWithPath: – initDirectoryWithFileWrappers: – initRegularFileWithContents: – initSymbolicLinkWithDestination: – initWithSerializedRepresentation:
Writing to a file or serializing	– writeToFile:atomically:updateFilenames: – serializedRepresentation
Checking a file wrapper's type	– isRegularFile – isDirectory – isSymbolicLink
Setting attributes	– setFilename: – filename – setPreferredFilename: – preferredFilename – setIcon: – icon – setFileAttributes: – fileAttributes
Updating	– needsToBeUpdatedFromPath: – updateFromPath:

Modifying a directory wrapper	– addFileWrapper: – removeFileWrapper: – addFileWithPath: – addRegularFileWithContents:preferredFilename: – addSymbolicLinkWithDestination:preferredFilename: – fileWrappers – keyForFileWrapper:
Inspecting a regular file wrapper	– regularFileContents
Inspecting a link wrapper	– symbolicLinkDestination

Instance Methods

addFileWithPath:

– (NSString *)**addFileWithPath:**(NSString *)*path*

Adds a new file wrapper to the receiving directory wrapper. Initializes the new file wrapper with **initWithPath:** using *path* as the argument, then adds the new file wrapper by invoking **addFileWrapper:**. Returns the dictionary key used for the newly added file wrapper within the directory wrapper. Raises an NSInternalInconsistencyException if sent to a regular file or link wrapper.

See also: – **addRegularFileWithContents:preferredFilename:**,
– **addSymbolicLinkWithDestination:preferredFilename:**, – **removeFileWrapper:**,
– **fileWrappers**

addFileWrapper:

– (NSString *)**addFileWrapper:**(NSFileWrapper *)*wrapper*

Adds *wrapper* to the receiving directory wrapper. Returns the dictionary key used for *wrapper* within the directory wrapper. Raises an NSInternalInconsistencyException if sent to a regular file or link wrapper, or an NSInvalidArgumentException if *wrapper* doesn't have a preferred name (set using **setPreferredName:**).

See also: – **addFileWithPath:**, – **addRegularFileWithContents:preferredFilename:**,
– **addSymbolicLinkWithDestination:preferredFilename:**, – **removeFileWrapper:**,
– **fileWrappers**

addRegularFileWithContents:preferredFilename:

– (NSString *)**addRegularFileWithContents:**(NSData *)*contents*
preferredFilename:(NSString *)*filename*

Adds a new regular file wrapper to the receiving directory wrapper. Initializes the new file wrapper with **initRegularFileWithContents:** using *contents* as the argument, sets its preferred name with **setPreferredName:** using *filename* as the argument, then adds the new file wrapper by invoking **addFileWrapper:**. Returns the dictionary key used for the newly added file wrapper within the directory wrapper. Raises an NSInternalInconsistencyException if sent to a regular file or link wrapper, or an NSInvalidArgumentException if *filename* is **nil** or empty.

See also: – **addFileWithPath:**, – **addSymbolicLinkWithDestination:preferredFilename:**, – **removeFileWrapper:**, – **fileWrappers**

addSymbolicLinkWithDestination:preferredFilename:

– (NSString *)**addSymbolicLinkWithDestination:**(NSString *)*path*
preferredFilename:(NSString *)*filename*

Adds a new link wrapper to the receiving directory wrapper. Initializes the new link wrapper with **initSymbolicLinkWithDestination:** using *path* as the argument, sets its preferred name with **setPreferredName:** using *filename* as the argument, then adds the new link wrapper by invoking **addFileWrapper:**. Returns the dictionary key used for the newly added link wrapper within the directory wrapper. Raises an NSInternalInconsistencyException if sent to a regular file or link wrapper, or an NSInvalidArgumentException if *filename* is **nil** or empty.

See also: – **addFileWithPath:**, – **addFileWrapper:**, – **addRegularFileWithContents:preferredFilename:**, – **removeFileWrapper:**, – **fileWrappers**

fileAttributes

– (NSDictionary *)**fileAttributes**

Returns the file attributes last read from disk or set using **setFileAttributes:**. These attributes are used whenever the file wrapper is saved using **writeToFile:atomically:updateFilenames:**. See the NSFileManager class specification for information on the contents of the attributes dictionary.

filename

– (NSString *)**filename**

Returns the filename for the last known disk representation of the receiver, or **nil** if the receiver has no filename. The filename is used for record-keeping purposes only, and is set automatically when the file

wrapper is created from disk using **initWithPath:** and when it's saved to a disk using **writeToFile:atomically:updateFilenames:** (although this method allows you to request that the filename not be updated).

See also: – **preferredFilename:**, – **setFilename:**

fileWrappers

– (NSDictionary *)**fileWrappers**

Returns the file wrappers contained in a directory wrapper. Raises an NSInternalInconsistencyException if sent to a regular file or link wrapper. See "Working with Directory Wrappers" in the class description for information on the dictionary.

See also: – **filename**, – **addFileWrapper:**

icon

– (NSImage *)**icon**

Returns an image that can be used to represent the file wrapper to the user, or **nil** if the file wrapper has none. You don't have to use this image; for example, a file viewer typically looks up icons automatically based on file extensions, and so wouldn't need this image. Similarly, if a file wrapper represents an image file, you can display the image directly rather than a file icon.

See also: – **setIcon:**

initDirectoryWithFileWrappers:

– (id)**initDirectoryWithFileWrappers:**(NSDictionary *)*wrappers*

Initializes a newly allocated NSFileWrapper as a directory wrapper containing *wrappers*. The new directory wrapper has no filename or associated disk representation until you save it using **writeToFile:atomically:updateFilenames:**. It's also initialized with open permissions; anyone can read, write, or change directory to the disk representations that it saves.

If any file wrapper in *wrappers* doesn't have a preferred name, its preferred name is automatically set to its corresponding dictionary key in *wrappers*.

This method is a designated initializer for the NSFileWrapper class. Returns **self**.

See also: – **setPreferredFilename:**, – **filename**, – **setFileAttributes:**

initRegularFileWithContents:

– (id)**initRegularFileWithContents:**(NSData *)*contents*

Initializes a newly allocated NSFileWrapper as a regular file wrapper with *contents*. The new file wrapper has no filename or associated disk representation until you save it using **writeToFile:atomically:updateFilenames:**. It's also initialized with open permissions; anyone can read or write the disk representations that it saves.

This method is a designated initializer for the NSFileWrapper class. Returns **self**.

See also: – **setPreferredFilename:**, – **filename**, – **fileAttributes**

initSymbolicLinkWithDestination:

– (id)**initSymbolicLinkWithDestination:**(NSString *)*path*

Initializes a newly allocated NSFileWrapper as a link wrapper pointing to *path*. The new file wrapper has no filename or associated disk representation until you save it using **writeToFile:atomically:updateFilenames:**. It's also initialized with open permissions; anyone can read or write the disk representations that it saves.

This method is a designated initializer for the NSFileWrapper class. Returns **self**.

See also: – **setPreferredFilename:**, – **filename**, – **fileAttributes**

initWithPath:

– (id)**initWithPath:**(NSString *)*path*

Initializes a newly allocated NSFileWrapper with the file or directory at *path*, setting its type to regular file, directory, or link wrapper based on the type of that file and caching the file's attributes. Also sets the receiver's preferred filename and recorded filename to the last component of *path*. If *path* identifies a directory, this method recursively creates file wrappers for each file or directory within that directory.

This method is a designated initializer for the NSFileWrapper class. Returns **self**.

See also: – **setPreferredFilename:**, – **filename**, – **fileAttributes**

initWithSerializedRepresentation:

– (id)**initWithSerializedRepresentation:**(NSData *)*data*

Initializes a newly allocated NSFileWrapper with *data*, setting its type to regular file, directory, or link wrapper based on the nature of that data and reading the file attributes from the data as well. *data* is a serialized representation of a file's or directory's contents in the format used for the pasteboard type

NSFileContentsPboardType. Data of this format is returned by such methods as **serializedRepresentation** or NSAttributedString's **RTFDFromRange:**.

The new file wrapper has no filename or associated disk representation until you save it using **writeToFile:atomically:updateFilenames:**. This method is a designated initializer for the NSFileWrapper class. Returns **self**.

See also: – **setPreferredFilename:**, – **filename**, – **fileAttributes**

isDirectory

– (BOOL)**isDirectory**

Returns YES if the receiver is a directory wrapper, NO otherwise.

See also: – **isRegularFile**, – **isSymbolicLink**

isRegularFile

– (BOOL)**isRegularFile**

Returns YES if the receiver is a regular file wrapper, NO otherwise.

See also: – **isDirectory**, – **isSymbolicLink**

isSymbolicLink

– (BOOL)**isSymbolicLink**

Returns YES if the receiver is a link wrapper, NO otherwise.

See also: – **isDirectory**, – **isRegularFile**

keyForFileWrapper:

– (NSString *)**keyForFileWrapper:**(NSFileWrapper *)*wrapper*

Returns the key by which the receiving directory wrapper stores *wrapper* in its dictionary (as returned by the **fileWrappers** method). This is not necessarily the filename for *wrapper*. Raises an NSInternalInconsistencyException if sent to a regular file or link wrapper.

See also: – **filename**

needsToBeUpdatedFromPath:

– (BOOL)**needsToBeUpdatedFromPath:**(NSString *)*path*

Returns YES if the receiver's contents on disk may have changed, NO otherwise. For a regular file wrapper, this is determined by comparing the modification time and access permissions of the file or directory at *path* against those of the receiver. For a link wrapper, this is determined by checking whether the destination path has changed (not by checking the modification time or access attributes of the destination). For a directory, this is determined as needed recursively for each file wrapper contained in the directory; added or removed files also count as changes.

See also: – **updateFromPath:**, – **fileAttributes**

preferredFilename

– (NSString *)**preferredFilename**

Returns the file wrapper's preferred filename. This name is used as the default dictionary key and filename when a file wrapper is added to a directory wrapper. However, if another file wrapper with the same preferred name already exists in the directory wrapper when the receiver is added, the dictionary key and filename assigned may differ from the preferred filename.

See also: – **setPreferredFilename:**, – **filename**

regularFileContents

– (NSData *)**regularFileContents**

Returns the contents of the receiving regular file wrapper. Raises an NSInternalInconsistencyException if sent to a directory or link wrapper.

removeFileWrapper:

– (void)**removeFileWrapper:**(NSFileWrapper *)*wrapper*

Removes *wrapper* from the receiving directory wrapper and releases it. Raises an NSInternalInconsistencyException if sent to a regular file or link wrapper.

See also: – **addFileWithPath:**, – **addFileWrapper:**,
– **addRegularFileWithContents:preferredFilename:**,
– **addSymbolicLinkWithDestination:preferredFilename:**, – **fileWrappers**

serializedRepresentation

– (NSData *)**serializedRepresentation**

Returns the receiver's contents as an opaque collection of data, in the format used for the pasteboard type NSFileContentsPboardType.

See also: – initWithSerializedRepresentation:

setFileAttributes:

– (void)**setFileAttributes:**(NSDictionary *)*attributes*

Sets the file attributes that are applied whenever the file wrapper is saved using **writeToFile:atomically:updateFilenames:** to *attributes.* See the NSFileManager class specification for information on the contents of the attributes dictionary.

See also: – **fileAttributes**

setFilename:

– (void)**setFilename:**(NSString *)*filename*

Sets the filename for the disk representation of the receiver to *filename*. The filename is used for record-keeping purposes only, and is set automatically when the file wrapper is saved to a disk using **writeToFile:atomically:updateFilenames:** (although this method allows you to request that the filename not be updated). You should rarely need to invoke this method.

Raises an NSInvalidArgumentException if *filename* is **nil** or empty.

See also: – **setPreferredFilename:**, – **filename**

setIcon:

– (void)**setIcon:**(NSImage *)*anImage*

Sets the image that can be used to represent the file wrapper to the user to *anImage*. You don't have to use this image; for example, a file viewer typically looks up icons automatically based on file extensions, and so wouldn't need this image. Similarly, if a file wrapper represents an image file, you can display the image directly rather than a file icon.

See also: – **icon**

setPreferredFilename:

– (void)**setPreferredFilename:**(NSString *)*filename*

Sets the receiver's preferred filename to *filename*. This name is used as the default dictionary key and filename when a file wrapper is added to a directory wrapper. However, if another file wrapper with the same preferred name already exists in the directory wrapper when the receiver is added, the dictionary key and filename assigned may differ from the preferred filename. Raises an NSInvalidArgumentException if *filename* is **nil** or empty.

See also: – **preferredFilename**, – **addFileWrapper:**, – **setFilename:**

symbolicLinkDestination

– (NSString *)**symbolicLinkDestination**

Returns the actual path represented by the receiving link wrapper. Raises NSInternalInconsistencyException if sent to a regular file or directory wrapper.

updateFromPath:

– (BOOL)**updateFromPath:**(NSString *)*path*

Re-reads the file wrapper's information from the file or directory at *path*, including contents or link destination, icon, file attributes. For a directory wrapper, the contained file wrappers are also sent **updateFromPath:** messages. If files in the directory on disk have been added or removed, corresponding file wrappers are released or created as needed. Returns YES if updating actually occurred, NO if it wasn't necessary.

See also: – **needsUpdateFromPath:**, – **updateAttachmentsFromPath:** (NSAttributedString class cluster)

writeToFile:atomically:updateFilenames:

– (BOOL)**writeToFile:**(NSString *)*path*
atomically:(BOOL)*atomicFlag*
updateFilenames:(BOOL)*updateNamesFlag*

Writes the receiver's contents to a file or directory at *path*. Returns YES on success and NO on failure. If *atomicFlag* is YES, attempts to write the file safely so that an existing file at *path* is not overwritten, nor does a new file at *path* actually get created, unless the write is successful. If *updateNamesFlag* is YES and the contents are successfully written, changes the receiver's filename to the last component of *path*, and the filenames of any children of a directory wrapper to the filenames under which they're written to disk.

If you're executing a "save" or "save as" style operation, pass YES for *updateNamesFlag;* if you're executing a "save to" style operation, pass NO for *updateNamesFlag*.

See also: – **filename**