# NSDocument Class Reference

**Cocoa > Design Guidelines**

**2009-01-06**

# Contents

# NSDocument Class Reference

| | |
|---|---|
| **Inherits from** | NSObject |
| **Conforms to** | NSUserInterfaceValidations<br>NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AppKit.framework |
| **Availability** | Available in Mac OS X v10.0 and later. |
| **Companion guide** | Document-Based Applications Overview |
| **Declared in** | NSDocument.h<br>NSDocumentScripting.h |
| **Related sample code** | iSpend<br>QTAudioExtractionPanel<br>QTKitPlayer<br>Simple Bindings Adoption<br>Sketch-112 |

## Class at a Glance

`NSDocument` is an abstract class that defines the interface for documents, objects that can internally represent data displayed in windows and that can read data from and write data to files. Documents create and manage one or more window controllers and are in turn managed by a document controller. Documents respond to first-responder action messages to save, revert, and print their data.

### Principal Attributes

- Window controllers
- Filenames
- Document types
- Print information

`init` (page 30)
 Designated initializer for new documents

`initWithContentsOfURL:ofType:error:` (page 32)
 For existing documents

## Commonly Used Methods

`dataOfType:error:` (page 22)
    Returns the document's data in a specified type.

`readFromData:ofType:error:` (page 40)
    Sets the contents of this document by reading from data of a specified type.

`writeToURL:ofType:error:` (page 64)
    Writes the document's data to a URL.

`readFromURL:ofType:error:` (page 42)
    Reads the document's data from a file.

`windowNibName` (page 62)
    Returns the name of the document's sole nib file (resulting in the creation of a window controller for the window in that file).

`makeWindowControllers` (page 34)
    Creates and returns the window controllers used to manage document windows.

## Overview

`NSDocument` is an abstract class that defines the interface for documents.

Conceptually, a document is a container for a body of information identified by a name under which it is stored in a disk file. In this sense, however, the document is not the same as the file but is an object in memory that owns and manages the document data. In the context of the Application Kit, a document is an instance of a custom `NSDocument` subclass that knows how to represent internally, in one or more formats, persistent data that is displayed in windows.

A document can read that data from a file and write it to a file. It is also the first-responder target for many menu commands related to documents, such as Save, Revert, and Print. A document manages its window's edited status and is set up to perform undo and redo operations. When a window is closing, the document is asked before the window delegate to approve the closing.

`NSDocument` is one of the triad of Application Kit classes that establish an architectural basis for document-based applications (the others being `NSDocumentController` and `NSWindowController`).

### Subclassing NSDocument

`NSDocument` is designed to be subclassed. That is, `NSDocument` is an abstract class, and your application must create at least one `NSDocument` subclass to use the document architecture. To create a useful `NSDocument` subclass, you must override some methods, and you can optionally override others.

The `NSDocument` class itself knows how to handle document data as undifferentiated lumps; although it understands that these lumps are typed, it knows nothing about particular types. In their overrides of the data-based reading and writing methods, subclasses must add the knowledge of particular types and how data of the document's native type is structured internally. Subclasses are also responsible for the creation of the window controllers that manage document windows and for the implementation of undo and redo. The `NSDocument` class takes care of much of the rest, including generally managing the state of the document.

See “Creating a Subclass of NSDocument” in *Document-Based Applications Overview* for more information about creating subclasses of `NSDocument`, particularly the list of primitive methods that subclasses must override and those that you can optionally override.

## Writing of HFS Creator and File Type Codes

The `fileAttributesToWriteToFile:ofType:saveOperation:` (page 70) method can be overridden to specify that a creator code or file type code (or both) should be written to a file as it is being saved. See `NSFileManager` for descriptions of the `NSFileHFSCreatorCode` and `NSFileHFSTypeCode` file attributes. The `NSDocument` implementation of `fileAttributesToWriteToFile:ofType:saveOperation:` returns zeroed-out creator and file type codes, effectively excluding creator code and file type code from the attribute preservation described in `fileAttributesToWriteToFile:ofType:saveOperation:`.

## NSDocument Saving Behavior

`NSDocument` implements document saving in a way that preserves, when possible, various attributes of each document, including:

- Creation date
- Permissions/privileges
- Location of the document’s icon in its parent folder’s Icon View Finder window
- Value of the document’s Show Extension setting

Care is also taken to save documents in a way that does not break any user-created aliases that may point to documents. As a result, some methods in any class of `NSDocument` may be invoked with parameters that do not have the same meaning as they did in early releases of Mac OS X. It is important that overrides of `writeToURL:ofType:error:` (page 64) and `writeToURL:ofType:forSaveOperation:originalContentsURL:error:` (page 65) make no assumptions about the file paths passed as parameters, including:

- The location to which the file is being written. This location might be a hidden temporary directory.
- The name of the file being written. It is possible that this file has no obvious relation to the document name.
- The relation of any file being passed, including the original file, to the return value of `fileName` (page 70).

# Tasks

### Initializing

`- init` (page 30)
    Initializes and returns an empty `NSDocument` object.

- `initWithContentsOfURL:ofType:error:` (page 32)
  Initializes a document located by a URL of a specified type.
- `initForURL:withContentsOfURL:ofType:error:` (page 31)
  Initializes a document located by a URL of a specified type, but by reading the contents for the document from a different URL.
- `initWithType:error:` (page 33)
  Initializes a document of a specified type.

## Loading Document Data

- `dataOfType:error:` (page 22)
  Creates and returns a data object that contains the contents of the document, formatted to a specified type.
- `fileWrapperOfType:error:` (page 28)
  Creates and returns a file wrapper that contains the contents of the document, formatted to the specified type.
- `readFromData:ofType:error:` (page 40)
  Sets the contents of this document by reading from data of a specified type and returns `YES` if successful.

## Creating and Managing Window Controllers

- `makeWindowControllers` (page 34)
  Subclasses may override this method to create the initial window controller(s) for the document.
- `windowNibName` (page 62)
  Overridden by subclasses to return the name of the document’s sole nib file.
- `windowControllerDidLoadNib:` (page 60)
  Sent after the specified window controller loads a nib file if the receiver is the nib file's owner.
- `windowControllerWillLoadNib:` (page 61)
  Sent before the specified window controller loads a nib file if the receiver is the nib file's owner.
- `windowControllers` (page 60)
  Returns the receiver’s current window controllers.
- `addWindowController:` (page 19)
  Adds the specified window controller to the array of window controllers associated with the receiver.
- `removeWindowController:` (page 42)
  Removes the specified window controller from the receiver’s array of window controllers.
- `shouldCloseWindowController:delegate:shouldCloseSelector:contextInfo:` (page 56)
  Invokes *`shouldCloseSelector`* with the result of `canCloseDocumentWithDelegate:shouldCloseSelector:contextInfo:` (page 21) if the the specified window controller that is closing is the last one or is marked as causing the document to close.

## Managing Document Windows

- `showWindows` (page 57)
  Displays all of the document's windows, bringing them to the front and making them main or key as necessary.
- `displayName` (page 23)
  Returns the name of the receiver as displayed in the title bars of the document's windows and in alert dialogs related to the document.
- `setWindow:` (page 54)
  Sets the `window` Interface Builder outlet of this class.
- `windowForSheet` (page 61)
  Returns the most appropriate window, of the windows associated with the receiver, to use as the parent window of a document-modal sheet.

## Reading From and Writing to Files

- `readFromFileWrapper:ofType:error:` (page 41)
  Sets the contents of this document by reading from a file wrapper of a specified type.
- `fileModificationDate` (page 25)
  Returns the last known modification date of the document's on-disk representation.
- `setFileModificationDate:` (page 51)
  Sets the last known modification date of the document's on-disk representation to the given modification date.
- `runModalSavePanelForSaveOperation:delegate:didSaveSelector:contextInfo:` (page 45)
  Presents a modal Save panel to the user, then tries to save the document if the user approves the panel.
- `shouldRunSavePanelWithAccessoryView` (page 56)
  Returns `YES` by default; as a result, when `NSDocument` displays the Save panel, it includes an accessory view containing a pop-up menu of supported writable document types.
- `keepBackupFile` (page 34)
  Returns whether the receiver should keep the backup files created before document data is written to a file (`NO` by default).

## Reading From and Writing to URLs

- `readFromURL:ofType:error:` (page 42)
  Sets the contents of this document by reading from a file or file package, of a specified type, located by a URL.
- `writeToURL:ofType:error:` (page 64)
  Writes the contents of the document to a file or file package located by a URL, formatted to a specified type.
- `writeSafelyToURL:ofType:forSaveOperation:error:` (page 63)
  Writes the contents of the document to a file or file package located by a URL.
- `writeToURL:ofType:forSaveOperation:originalContentsURL:error:` (page 65)
  Writes the contents of the document to a file or file package located by a URL.

- `setFileURL:` (page 52)
  Sets the location of the document's on-disk representation.
- `fileURL` (page 27)
  Returns the location of the document's on-disk representation.
- `fileAttributesToWriteToURL:ofType:forSaveOperation:originalContentsURL:error:` (page 24)
  As a file is being saved, returns the attributes that should be written to a file or file package located by a URL, formatted to a specified type, for a particular kind of save operation.
- `saveToURL:ofType:forSaveOperation:delegate:didSaveSelector:contextInfo:` (page 49)
  Saves the contents of the document to a file or file package located by a URL, formatted to a specified type, for a particular kind of save operation.
- `saveToURL:ofType:forSaveOperation:error:` (page 50)
  Saves the contents of the document to a file or file package located by a URL, formatted to a specified type, for a particular kind of save operation, and returns `YES` if successful.

## Autosaving

- `hasUnautosavedChanges` (page 30)
  Return `YES` if the document has changes that have not been autosaved, as determined by the history of previous invocations of `updateChangeCount:` (page 58).
- `autosaveDocumentWithDelegate:didAutosaveSelector:contextInfo:` (page 20)
  Autosaves the document's contents at an appropriate location.
- `autosavingFileType` (page 20)
  Returns the document type that should be used for an autosave operation.
- `setAutosavedContentsFileURL:` (page 50)
  Sets the location of the most recently autosaved document contents.
- `autosavedContentsFileURL` (page 19)
  Returns the location of the most recently autosaved document contents.

## Managing Document Status

- `isDocumentEdited` (page 33)
  Returns `YES` if the receiver has changes that have not been saved, `NO` otherwise.
- `updateChangeCount:` (page 58)
  Updates the receiver's change count according to the given change type.
- `fileNameExtensionWasHiddenInLastRunSavePanel` (page 26)
  Returns `YES` if a Save panel was presented by this document and the user chose to hide the name extension of the file that was selected in that Save panel.

## Handling User Actions

– `prepareSavePanel:` (page 36)
  Invoked by `runModalSavePanelForSaveOperation:delegate:didSaveSelector:contextInfo:` (page 45) to do any customization of the given Save panel.

– `printDocument:` (page 38)
  Prints the receiver in response to the user choosing the Print menu command.

– `runPageLayout:` (page 46)
  The action method invoked in the receiver as first responder when the user chooses the Page Setup menu command.

– `revertDocumentToSaved:` (page 43)
  The action of the File menu item Revert in a document-based application.

– `saveDocument:` (page 47)
  The action method invoked in the receiver as first responder when the user chooses the Save menu command.

– `saveDocumentAs:` (page 47)
  The action method invoked in the receiver as first responder when the user chooses the Save As menu command.

– `saveDocumentTo:` (page 48)
  The action method invoked in the receiver as first responder when the user chooses the Save To menu command.

– `saveDocumentWithDelegate:didSaveSelector:contextInfo:` (page 48)
  Saves the document.

## Closing Documents

– `canCloseDocumentWithDelegate:shouldCloseSelector:contextInfo:` (page 21)
  If the receiver is not dirty, this method immediately calls the *`shouldCloseSelector`* callback on the specified delegate with `YES`.

– `close` (page 22)
  Closes all windows owned by the receiver and removes the receiver from the list of documents maintained by the document controller, which consequently releases it.

## Reverting Documents

– `revertToContentsOfURL:ofType:error:` (page 43)
  Discards all unsaved document modifications and replaces the document's contents by reading a file or file package located by a URL of a specified type.

## Printing Documents

– `printInfo` (page 39)
  Returns the receiver's customized `NSPrintInfo` object or the default `NSPrintInfo` instance.

- `setPrintInfo:` (page 53)
  Sets the receiver's `NSPrintInfo` object.
- `preparePageLayout:` (page 35)
  Invoked by `runModalPageLayoutWithPrintInfo:` (page 75) and `runModalPageLayoutWithPrintInfo:delegate:didRunSelector:contextInfo:` (page 44) to do any customization of the Page Layout panel *`pageLayout`*, such as adding an accessory view.
- `runModalPageLayoutWithPrintInfo:delegate:didRunSelector:contextInfo:` (page 44)
  Runs the modal page layout panel with the receiver's printing information object
- `runModalPrintOperation:delegate:didRunSelector:contextInfo:` (page 45)
  Runs the specified print operation modally.
- `shouldChangePrintInfo:` (page 55)
  Returns a Boolean value indicating whether the receiver should allow changes to the default `NSPrintInfo` object used in printing the document.
- `printDocumentWithSettings:showPrintPanel:delegate:didPrintSelector:contextInfo:` (page 38)
  Prints the document.
- `printOperationWithSettings:error:` (page 40)
  Creates a print operation and returns it if successful.

## Handling Errors

- `presentError:modalForWindow:delegate:didPresentSelector:contextInfo:` (page 37)
  Presents an error alert to the user as a modal panel.
- `presentError:` (page 36)
  Presents an error alert to the user as a modal panel.
- `willPresentError:` (page 59)
  Called when the receiver is about to present an error.

## Working with Undo Manager

- `hasUndoManager` (page 30)
  Returns a Boolean value indicating whether the receiver owns or should own an `NSUndoManager` object.
- `setHasUndoManager:` (page 52)
  Sets whether the receiver has its own `NSUndoManager` object.
- `setUndoManager:` (page 54)
  Sets the undo manager owned by the receiver to the specified undo manager and releases any undo manager currently owned by the receiver.
- `undoManager` (page 57)
  Returns the receiver's undo manager.

## Managing File Types

- `setFileType:` (page 51)
  Sets the document type under which the file is saved.
- `fileType` (page 26)
  Returns the document type under which the receiver is saved.
- `fileTypeFromLastRunSavePanel` (page 27)
  Returns the file type that was last selected in the Save panel.

+ `isNativeType:` (page 17)
  Returns a Boolean value indicating whether document data of the specified type is a native type—one the receiver can both read and write.

+ `readableTypes` (page 18)
  Returns the types of data the receiver can read natively and any types filterable to that native type.

+ `writableTypes` (page 18)
  Returns the types of data the receiver can write natively and any types filterable to that native type.

- `writableTypesForSaveOperation:` (page 62)
  Returns the names of the types to which this document can be saved for a specified kind of save operation.
- `fileNameExtensionForType:saveOperation:` (page 25)
  Returns a filename extension that can be appended to a base filename, for a specified file type and kind of save operation.

## Validating User Interface Items

- `validateUserInterfaceItem:` (page 59)
  Validates the specified user interface item that the receiver manages.

## Scripting

- `handleCloseScriptCommand:` (page 29)
  Handles the Close AppleScript command by attempting to close the document.
- `handlePrintScriptCommand:` (page 29)
  Handles the Print AppleScript command by attempting to print the document.
- `handleSaveScriptCommand:` (page 29)
  Handles the Save AppleScript command by attempting to save the document.
- `objectSpecifier` (page 35)
  Returns an object specifier for the document.
- `lastComponentOfFileName` (page 34)
  Returns the document name in terms of the scripting name property (the name a script writer would use to specify the document in a script).
- `setLastComponentOfFileName:` (page 53)
  Sets the document name to the given string in terms of the scripting name property (the name a script writer would use to specify the document in a script).

## Deprecated Methods

- `validateMenuItem:` (page 58)
  Validates the Revert menu item and items selected from the Save panel's pop-up list of writable document types items. (Deprecated. Use `validateUserInterfaceItem:` (page 59) instead.)
- `canCloseDocument` (page 21)
  This method is no longer supported. (Deprecated. Use `canCloseDocumentWithDelegate:shouldCloseSelector:contextInfo:` (page 21) instead.)
- `fileNameFromRunningSavePanelForSaveOperation:` (page 26)
  Returns the filename entered into the Save panel. (Deprecated. Use `saveDocumentWithDelegate:didSaveSelector:contextInfo:` (page 48) instead.)
- `shouldCloseWindowController:` (page 55)
  Gives the user an opportunity to save the document. (Deprecated. Use `shouldCloseWindowController:delegate:shouldCloseSelector:contextInfo:` (page 56) instead.)
- `dataRepresentationOfType:` (page 69) Deprecated in Mac OS X v10.4
  A primitive method overridden by subclasses to return a data object that represents the data of the receiver in a given type. (Deprecated. Use `dataOfType:error:` (page 22) instead.)
- `fileAttributesToWriteToFile:ofType:saveOperation:` (page 70) Deprecated in Mac OS X v10.4
  Returns the file attributes that should be written to the named document file of the specified type. (Deprecated. Use `fileAttributesToWriteToURL:ofType:forSaveOperation:originalContentsURL:error:` (page 24) instead.)
- `fileName` (page 70) Deprecated in Mac OS X v10.4
  Returns the filename (as a fully qualified path) under which the receiver has been saved. (Deprecated. Use `fileURL` (page 27) instead.)
- `fileWrapperRepresentationOfType:` (page 70) Deprecated in Mac OS X v10.4
  Returns an NSFileWrapper object that represents the data of the receiver in a given type. (Deprecated. Use `fileWrapperOfType:error:` (page 28) instead.)
- `initWithContentsOfFile:ofType:` (page 71) Deprecated in Mac OS X v10.4
  Initializes and returns an `NSDocument` object. (Deprecated. Use `initWithContentsOfURL:ofType:error:` (page 32) instead.)
- `initWithContentsOfURL:ofType:` (page 71) Deprecated in Mac OS X v10.4
  Initializes and returns an NSDocument object of a given document type. (Deprecated. Use `initWithContentsOfURL:ofType:error:` (page 32) instead.)
- `loadDataRepresentation:ofType:` (page 72) Deprecated in Mac OS X v10.4
  Overridden by subclasses to load document data. (Deprecated. Use `readFromData:ofType:error:` (page 40) instead.)
- `loadFileWrapperRepresentation:ofType:` (page 73) Deprecated in Mac OS X v10.4
  Loads document data from a given file wrapper. (Deprecated. Use `readFromFileWrapper:ofType:error:` (page 41) instead.)
- `printShowingPrintPanel:` (page 73) Deprecated in Mac OS X v10.4
  Overridden by subclasses to print the current document's (the receiver's) data. (Deprecated. Use `printDocumentWithSettings:showPrintPanel:delegate:didPrintSelector:contextInfo:` (page 38) instead.)

- `readFromFile:ofType:` (page 74) Deprecated in Mac OS X v10.4
  Reads and loads document data of the given type from the given file. (Deprecated. Use `readFromURL:ofType:error:` (page 42) instead.)
- `readFromURL:ofType:` (page 74) Deprecated in Mac OS X v10.4
  Reads and loads document data. (Deprecated. Use `readFromURL:ofType:error:` (page 42) instead.)
- `revertToSavedFromFile:ofType:` (page 75) Deprecated in Mac OS X v10.4
  Reverts the receiver to the data stored in the file system. (Deprecated. Use `revertToContentsOfURL:ofType:error:` (page 43) instead.)
- `revertToSavedFromURL:ofType:` (page 75) Deprecated in Mac OS X v10.4
  Reverts the receiver. (Deprecated. Use `revertToContentsOfURL:ofType:error:` (page 43) instead.)
- `runModalPageLayoutWithPrintInfo:` (page 75) Deprecated in Mac OS X v10.4
  Runs the page layout modal panel with the receiver's printing information object. (Deprecated. Use `runModalPageLayoutWithPrintInfo:delegate:didRunSelector:contextInfo:` (page 44) instead.)
- `saveToFile:saveOperation:delegate:didSaveSelector:contextInfo:` (page 76) Deprecated in Mac OS X v10.4
  Called after the user has been given the opportunity to select a destination through the modal Save panel. (Deprecated. Use `saveToURL:ofType:forSaveOperation:delegate:didSaveSelector:contextInfo:` (page 49) instead.)
- `setFileName:` (page 77) Deprecated in Mac OS X v10.4
  Sets the file (filename and directory path) under which document data is saved. (Deprecated. Use `setFileURL:` (page 52) instead.)
- `writeToFile:ofType:` (page 77) Deprecated in Mac OS X v10.4
  Writes document data to a file. (Deprecated. Use `writeToURL:ofType:error:` (page 64) instead.)
- `writeToFile:ofType:originalFile:saveOperation:` (page 78) Deprecated in Mac OS X v10.4
  Writes the receiver document's contents to a file. (Deprecated. Use `writeToURL:ofType:forSaveOperation:originalContentsURL:error:` (page 65) instead.)
- `writeToURL:ofType:` (page 78) Deprecated in Mac OS X v10.4
  Writes document data to a URL. (Deprecated. Use `writeToURL:ofType:error:` (page 64) instead.)
- `writeWithBackupToFile:ofType:saveOperation:` (page 78) Deprecated in Mac OS X v10.4
  This method is called by action methods to save document contents to a file. (Deprecated. Use `writeSafelyToURL:ofType:forSaveOperation:error:` (page 63) instead.)

## Class Methods

### isNativeType:

Returns a Boolean value indicating whether document data of the specified type is a native type—one the receiver can both read and write.

```
+ (BOOL)isNativeType:(NSString *)aType
```

**Parameters**

*aType*
: The string that identifies the document type to test.

**Return Value**

`YES` if the document type is a native type; otherwise, `NO`.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

`+ readableTypes` (page 18)
`+ writableTypes` (page 18)

**Declared In**

`NSDocument.h`

## readableTypes

Returns the types of data the receiver can read natively and any types filterable to that native type.

```
+ (NSArray *)readableTypes
```

**Return Value**

An array of `NSString` objects representing the readable document types.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

`+ isNativeType:` (page 17)
`+ writableTypes` (page 18)

**Related Sample Code**

iSpend

**Declared In**

`NSDocument.h`

## writableTypes

Returns the types of data the receiver can write natively and any types filterable to that native type.

```
+ (NSArray *)writableTypes
```

**Return Value**

An array of `NSString` objects representing the writable document types.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

`+ isNativeType:` (page 17)

`+ readableTypes` (page 18)

**Related Sample Code**
iSpend

**Declared In**
`NSDocument.h`

## Instance Methods

### addWindowController:

Adds the specified window controller to the array of window controllers associated with the receiver.

```
- (void)addWindowController:(NSWindowController *)aController
```

**Parameters**

*aController*
The window controller that is added.

**Discussion**
An `NSDocument` object uses this list when it displays all document windows, sets window edited status upon an undo or redo operation, and modifies window titles. The method also sets the document outlet of the window controller to `self` if it is not already set. If you create window controllers by overriding `windowNibName` (page 62), this method is invoked automatically. If you create window controllers in `makeWindowControllers` (page 34) or in any other context, such as in response to a user event, you should invoke this method for each created window controller. To remove a window controller from the list of active controllers, send it the `NSWindowController` message `close`.

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

**See Also**
`- setDocument:` (NSWindowController)

**Declared In**
`NSDocument.h`

### autosavedContentsFileURL

Returns the location of the most recently autosaved document contents.

```
- (NSURL *)autosavedContentsFileURL
```

**Return Value**
The location of the most recently autosaved document contents.

**Discussion**
The default implementation of this method just returns whatever was stored by a previous invocation of the default implementation of `setAutosavedContentsFileURL:` (page 50).

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

**See Also**
`- setAutosavedContentsFileURL:` (page 50)

**Declared In**
`NSDocument.h`

## autosaveDocumentWithDelegate:didAutosaveSelector:contextInfo:

Autosaves the document's contents at an appropriate location.

```
- (void)autosaveDocumentWithDelegate:(id)delegate
    didAutosaveSelector:(SEL)didAutosaveSelector contextInfo:(void *)contextInfo
```

**Parameters**

*delegate*
The delegate to which the selector message is sent.

*didAutosaveSelector*
The selector of the message sent to the delegate.

*contextInfo*
Object passed with the callback to provide any additional context information.

**Discussion**
After autosaving, sends the message selected by *didAutosaveSelector* to the delegate, with *contextInfo* as the last argument. The method selected by *didAutosaveSelector* must have the same signature as:

```
- (void)document:(NSDocument *)document didAutosave:(BOOL)didAutosaveSuccessfully
  contextInfo:(void *)contextInfo
```

If an error occurs while autosaving, the method reports it to the user before sending the delegate a `succeeded:NO` message.

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

**See Also**
`- autosavedContentsFileURL` (page 19)

**Declared In**
`NSDocument.h`

## autosavingFileType

Returns the document type that should be used for an autosave operation.

```
- (NSString *)autosavingFileType
```

**Return Value**
The string that identifies the document type.

**Discussion**
The default implementation just returns `[self fileType]`. You can override this method and return `nil` in your override to completely disable autosaving of individual documents (because `NSDocumentController` does not send `autosaveDocumentWithDelegate:didAutosaveSelector:contextInfo:` to a document that has no autosaving file type). You can also override it if your application defines a document type that is specifically designed for autosaving, for example, one that efficiently represents document content changes instead of complete document contents.

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

**Declared In**
`NSDocument.h`

## canCloseDocument

This method is no longer supported. (**Deprecated.** Use `canCloseDocumentWithDelegate:shouldCloseSelector:contextInfo:` (page 21) instead.)

```
- (BOOL)canCloseDocument
```

**Availability**
Available in Mac OS X v10.0 through Mac OS X v10.3.

**Declared In**
`NSDocument.h`

## canCloseDocumentWithDelegate:shouldCloseSelector:contextInfo:

If the receiver is not dirty, this method immediately calls the *`shouldCloseSelector`* callback on the specified delegate with `YES`.

```
- (void)canCloseDocumentWithDelegate:(id)delegate
    shouldCloseSelector:(SEL)shouldCloseSelector contextInfo:(void *)contextInfo
```

**Parameters**

*`delegate`*
The delegate to which the selector message is sent.

*`shouldCloseSelector`*
The selector of the message sent to the delegate.

*`contextInfo`*
Object passed with the callback to provide any additional context information.

**Discussion**
If the receiver is dirty, an alert is presented giving the user a chance to save, not save, or cancel. If the user chooses to save, this method saves the document. If the save completes successfully, this method calls the callback with `YES`. If the save is canceled or otherwise unsuccessful, this method calls the callback with `NO`. This method may be called by `shouldCloseWindowController:delegate:shouldCloseSelector:contextInfo:` (page 56). It is also called by the `NSDocumentController` method `closeAllDocuments`. You should call it before you call `close` (page 22) if you are closing the document and want to give the user a chance to save any edits. Pass the *`contextInfo`* object with the callback.

The *shouldCloseSelector* callback method should have the following signature:

```
- (void)document:(NSDocument *)doc shouldClose:(BOOL)shouldClose
contextInfo:(void  *)contextInfo
```

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

**Declared In**
`NSDocument.h`

## close

Closes all windows owned by the receiver and removes the receiver from the list of documents maintained by the document controller, which consequently releases it.

```
- (void)close
```

**Discussion**
This method closes the document immediately, without asking users if they want to save the document.

This method may not always be called. Additional information on application termination can be found in Graceful Application Termination.

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

**See Also**
`- canCloseDocumentWithDelegate:shouldCloseSelector:contextInfo:` (page 21)
`- shouldCloseWindowController:delegate:shouldCloseSelector:contextInfo:` (page 56)

**Related Sample Code**
ThreadsExporter
ThreadsExportMovie
ThreadsImporter
ThreadsImportMovie

**Declared In**
`NSDocument.h`

## dataOfType:error:

Creates and returns a data object that contains the contents of the document, formatted to a specified type.

```
- (NSData *)dataOfType:(NSString *)typeName error:(NSError **)outError
```

**Parameters**

*typeName*
The string that identifies the document type.

*outError*
On return, If the data object could not be created, a pointer to an error object that encapsulates the reason it could not be created.

**Return Value**
A data object containing the document contents, or, if the data object could not be created, `nil`.

**Discussion**
The default implementation of this method throws an exception because at least one of the writing methods (this method, `writeToURL:ofType:error:` (page 64), `fileWrapperOfType:error:` (page 28), or `writeToURL:ofType:forSaveOperation:originalContentsURL:error:` (page 65)) must be overridden.

For backward binary compatibility with Mac OS X v10.3 and earlier, the default implementation of this method instead invokes `dataRepresentationOfType:`*`typeName`* on `self` if `dataRepresentationOfType:` is overridden.

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

**See Also**
`– writeToURL:ofType:error:` (page 64)
`– fileWrapperOfType:error:` (page 28)

**Declared In**
`NSDocument.h`

## displayName

Returns the name of the receiver as displayed in the title bars of the document's windows and in alert dialogs related to the document.

```
- (NSString *)displayName
```

**Return Value**
The display name of the receiver.

**Discussion**
If the document has been saved, the display name is the last component of the directory location of the saved file (for example, "`MyDocument`" if the path is "`/tmp/MyDocument.rtf`"). If the document is new, `NSDocument` makes the display name "Untitled *n*," where *n* is a number in a sequence of new and unsaved documents. The displayable name also takes into account whether the document's filename extension should be hidden. Subclasses of `NSWindowController` can override `windowTitleForDocumentDisplayName:` to modify the display name as it appears in window titles.

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

**Related Sample Code**
EnhancedAudioBurn
QTMetadataEditor

**Declared In**
`NSDocument.h`

## fileAttributesToWriteToURL:ofType:forSaveOperation:originalContentsURL:error:

As a file is being saved, returns the attributes that should be written to a file or file package located by a URL, formatted to a specified type, for a particular kind of save operation.

```
- (NSDictionary *)fileAttributesToWriteToURL:(NSURL *)absoluteURL ofType:(NSString
   *)typeName forSaveOperation:(NSSaveOperationType)saveOperation
  originalContentsURL:(NSURL *)absoluteOriginalContentsURL error:(NSError
  **)outError
```

**Parameters**

*absoluteURL*
: The location to which the document is being written.

*typeName*
: The string that identifies the document type.

*saveOperation*
: The type of save operation.

*absoluteOriginalContentsURL*
: The location of the previously saved copy of the document (if not `nil`).

*outError*
: On return, If the attributes could not be returned, a pointer to an error object that encapsulates the reason they could not be returned.

**Return Value**

A dictionary containing the attributes to be written, or `nil` if unsuccessful.

**Discussion**

The set of valid file attributes is a subset of those understood by the `NSFileManager` class. The default implementation of this method returns a dictionary with `NSFileHFSCreatorCode` and `NSFileHFSTypeCode` entries that have a value of 0 for `NSSaveOperation`, or a dictionary with an appropriate `NSFileExtensionHidden` entry for `NSSaveAsOperation` and `NSSaveToOperation`. You can override this method to customize the attributes that are written to document files.

This method is meant to be used just for attributes that need to be written for the first time, for `NSSaveAsOperation` and `NSSaveToOperation`.

Invokers of this method should silently ignore invalid attributes. Of particular interest is the `NSFileExtensionHidden` attribute, which is documented in `NSFileManager`.

The dictionary returned by the default implementation of this method contains an `NSFileExtensionHidden` entry when that is appropriate. Your subclass of `NSDocument` can override this method to control the attributes that are set during a save operation. An override of this method should return a copy of the dictionary returned by its superclass's version of this method, with appropriate alterations.

An override of `writeSafelyToURL:ofType:forSaveOperation:error:` (page 63) should invoke this method and set the returned attributes on the written document file, possibly using the `NSFileManager` method `changeFileAttributes:atPath:`.

Implementers of overrides of this method should not assume that:

- The file pointed to by *absoluteURL* at the moment the method is invoked, if there is one, is related to the document itself. It may be an unrelated file that is about to be overwritten.
- The `fileURL` (page 27) or `fileType` (page 26) method will return anything useful at the moment.

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

**Declared In**
`NSDocument.h`

## fileModificationDate

Returns the last known modification date of the document's on-disk representation.

```
- (NSDate *)fileModificationDate
```

**Return Value**
The file modification date.

**Discussion**
The `NSDocument` default file saving machinery uses this information to warn the user when the on-disk representation of an open document has been modified by something other than the current application.

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

**See Also**
`- setFileModificationDate:` (page 51)

**Declared In**
`NSDocument.h`

## fileNameExtensionForType:saveOperation:

Returns a filename extension that can be appended to a base filename, for a specified file type and kind of save operation.

```
- (NSString *)fileNameExtensionForType:(NSString *)typeName
    saveOperation:(NSSaveOperationType)saveOperation
```

**Parameters**

*typeName*
The file type.

*saveOperation*
The kind of save operation.

**Return Value**
The filename extension.

**Discussion**
The default implementation of this method invokes `preferredFileNameExtensionForType:` on the shared workspace object if the type is a UTI or, if it is not, for backward binary compatibility with Mac OS X v10.4 and earlier, invokes `fileExtensionsFromType:` on the shared document controller and chooses the first filename extension in the returned array.

You can override this method to customize the appending of extensions to filenames by `NSDocument`. In Mac OS X v10.5, it's only invoked from two places in the Application Kit:

1. The `autosaveDocumentWithDelegate:didAutosaveSelector:contextInfo:` (page 20) method uses this method when creating a new filename for the autosaved contents.
2. The `handleSaveScriptCommand:` (page 29) method uses this method when adding an extension to the filename specified by a script.

In all other cases, the name of any file being saved will have been fully specified by the user with the Save panel (whether they know it or not).

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

**Declared In**
`NSDocument.h`

## fileNameExtensionWasHiddenInLastRunSavePanel

Returns `YES` if a Save panel was presented by this document and the user chose to hide the name extension of the file that was selected in that Save panel.

```
- (BOOL)fileNameExtensionWasHiddenInLastRunSavePanel
```

**Return Value**
`YES` if a Save panel was presented and the user chose to hide the extension; otherwise, `NO`.

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

**Declared In**
`NSDocument.h`

## fileNameFromRunningSavePanelForSaveOperation:

Returns the filename entered into the Save panel. (**Deprecated.** Use `saveDocumentWithDelegate:didSaveSelector:contextInfo:` (page 48) instead.)

```
- (NSString
    *)fileNameFromRunningSavePanelForSaveOperation:(NSSaveOperationType)saveOperation
```

**Availability**
Available in Mac OS X v10.0 through Mac OS X v10.3.

**Declared In**
`NSDocument.h`

## fileType

Returns the document type under which the receiver is saved.

```
- (NSString *)fileType
```

**Return Value**
The string that identifies the document type.

**Discussion**
When a document is saved, the type is determined by the entries in the application's information property list (specified in `Info.plist`).

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

**See Also**
`- setFileType:` (page 51)

**Related Sample Code**
ThreadsExporter
ThreadsImporter
ThreadsImportMovie

**Declared In**
`NSDocument.h`

## fileTypeFromLastRunSavePanel

Returns the file type that was last selected in the Save panel.

```
- (NSString *)fileTypeFromLastRunSavePanel
```

**Return Value**
The string that identifies the document type.

**Discussion**
This type is primarily used by the `saveDocument:` (page 47), `saveDocumentAs:` (page 47), and `saveDocumentTo:` (page 48) methods to determine the type the user chose after the Save panel has been run.

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

**Declared In**
`NSDocument.h`

## fileURL

Returns the location of the document's on-disk representation.

```
- (NSURL *)fileURL
```

**Return Value**
The document's location.

**Discussion**
The default implementation of this method returns whatever was stored by a previous invocation of the default implementation of `setFileURL:` (page 52). For backward binary compatibility with Mac OS X v10.3 and earlier, if `fileName` (page 70) is overridden, the default implementation of this method instead invokes `[self fileName]` and returns the result as a URL.

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

**See Also**
`- setFileURL:` (page 52)

**Related Sample Code**
CocoaSpeechSynthesisExample
Departments and Employees
iSpend
QTMetadataEditor

**Declared In**
`NSDocument.h`

## fileWrapperOfType:error:

Creates and returns a file wrapper that contains the contents of the document, formatted to the specified type.

```
- (NSFileWrapper *)fileWrapperOfType:(NSString *)typeName error:(NSError **)outError
```

**Parameters**

*typeName*
The string that identifies the document type.

*outError*
On return, If the file wrapper could not be created, a pointer to an error object that encapsulates the reason it could not be created.

**Return Value**
A file wrapper containing the document contents, or, if the file wrapper could not be created, `nil`.

**Discussion**
For backward binary compatibility with Mac OS X v10.3 and earlier, if `fileWrapperRepresentationOfType:` (page 70) is overridden, the default implementation of this method instead invokes `[self fileWrapperRepresentationOfType:typeName]`.

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

**See Also**
`- dataOfType:error:` (page 22)

**Declared In**
`NSDocument.h`

## handleCloseScriptCommand:

Handles the Close AppleScript command by attempting to close the document.

```
- (id)handleCloseScriptCommand:(NSCloseCommand *)command
```

**Parameters**

*command*

A Close AppleScript command object.

**Discussion**

Extracts Close command arguments from the `command` object and uses them to determine how to close the document—specifically, whether to ignore unsaved changes, save changes automatically, or ask the user and to identify the file in which to save the document (by default, the file that was opened or previously saved to). A Close AppleScript command may specify more than one document to close. If so, a message is sent to each document object.

**Availability**

Available in Mac OS X v10.0 and later.

**Declared In**

`NSDocumentScripting.h`

## handlePrintScriptCommand:

Handles the Print AppleScript command by attempting to print the document.

```
- (id)handlePrintScriptCommand:(NSScriptCommand *)command
```

**Parameters**

*command*

An AppleScript command object.

**Discussion**

Extracts Print command arguments from the `command` object and uses them to determine how to print the document—specifically, any print settings and whether to show the Print dialog. A Print AppleScript command may specify more than one document to print. If so, a message is sent to each document.

**Availability**

Available in Mac OS X v10.0 and later.

**Declared In**

`NSDocumentScripting.h`

## handleSaveScriptCommand:

Handles the Save AppleScript command by attempting to save the document.

```
- (id)handleSaveScriptCommand:(NSScriptCommand *)command
```

**Parameters**

*command*

An AppleScript command object.

**Discussion**
Extracts Save command arguments from the *`command`* object and uses them to determine the file in which to save the document and the file type.

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

**Declared In**
`NSDocumentScripting.h`

## hasUnautosavedChanges

Return `YES` if the document has changes that have not been autosaved, as determined by the history of previous invocations of `updateChangeCount:` (page 58).

```
- (BOOL)hasUnautosavedChanges
```

**Return Value**
`YES` if the document has changes that have not been autosaved; otherwise, `NO`.

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

**Declared In**
`NSDocument.h`

## hasUndoManager

Returns a Boolean value indicating whether the receiver owns or should own an `NSUndoManager` object.

```
- (BOOL)hasUndoManager
```

**Return Value**
`YES` if the receiver has its own undo manager; otherwise, `NO`.

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

**See Also**
`- setHasUndoManager:` (page 52)

**Declared In**
`NSDocument.h`

## init

Initializes and returns an empty `NSDocument` object.

```
- (id)init
```

**Return Value**
An initialized `NSDocument` object.

**Discussion**
This initializer (the designated initializer) is invoked by each of the other `NSDocument` initialization methods.

You can override this method to perform initialization that must be done both when creating new empty documents and when opening existing documents. Your override must invoke `super` to initialize private `NSDocument` instance variables. It must never return `nil`. If an error can occur during object initialization, check for the error in an override of `initWithType:error:` (page 33), `initWithContentsOfURL:ofType:error:` (page 32), or `initForURL:withContentsOfURL:ofType:error:` (page 31), because those methods can return `NSError` objects.

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

**Related Sample Code**
iSpend
QTKitAdvancedDocument
QTMetadataEditor
Simple Bindings Adoption
Sketch-112

**Declared In**
`NSDocument.h`

## initForURL:withContentsOfURL:ofType:error:

Initializes a document located by a URL of a specified type, but by reading the contents for the document from a different URL.

```
- (id)initForURL:(NSURL *)absoluteDocumentURL withContentsOfURL:(NSURL
    *)absoluteDocumentContentsURL ofType:(NSString *)typeName error:(NSError
    **)outError
```

**Parameters**

*absoluteDocumentURL*
The URL where the document is located.

*absoluteDocumentContentsURL*
The URL from which the contents of the document are obtained.

*typeName*
The string that identifies the document type.

*outError*
On return, If initialization is unsuccessful, a pointer to an error object that encapsulates the reason the document could not be created.

**Return Value**
The initialized `NSDocument` object, or, if the document could not be created, `nil`.

**Discussion**
The *absoluteDocumentURL* argument is `nil` if the initializing is part of the reopening of an autosaved document when the autosaved document was never explicitly saved.

During reopening of autosaved documents, this method uses the following `NSDocumentChangeType` constant to indicate that an autosaved document is being reopened:

`NSChangeReadOtherContents`

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

**Declared In**
`NSDocument.h`

## initWithContentsOfURL:ofType:error:

Initializes a document located by a URL of a specified type.

```
- (id)initWithContentsOfURL:(NSURL *)absoluteURL ofType:(NSString *)typeName
    error:(NSError **)outError
```

**Parameters**

*absoluteURL*
The URL from which the contents of the document are obtained.

*typeName*
The string that identifies the document type.

*outError*
On return, If initialization is unsuccessful, a pointer to an error object that encapsulates the reason the document could not be created.

**Return Value**
The initialized `NSDocument` object, or, if the document could not be created, `nil`.

**Discussion**
You can override this method to customize the reopening of autosaved documents.

This method is invoked by the `NSDocumentController` method `makeDocumentWithContentsOfURL:ofType:error:`. The default implementation of this method invokes `init` (page 30), `readFromURL:ofType:error:` (page 42), `setFileURL:` (page 52), `setFileType:` (page 51), and `setFileModificationDate:` (page 51).

For backward binary compatibility with Mac OS X v10.3 and earlier, the default implementation of this method instead invokes `initWithContentsOfFile:ofType:` (page 71) if it is overridden and the URL uses the `file:` scheme. It still invokes `setFileModificationDate:` in this situation.

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

**Related Sample Code**
QTKitCreateMovie
QTKitFrameStepper

**Declared In**
`NSDocument.h`

## initWithType:error:

Initializes a document of a specified type.

```
- (id)initWithType:(NSString *)typeName error:(NSError **)outError
```

**Parameters**

*typeName*

The string that identifies the document type.

*outError*

On return, If initialization is unsuccessful, a pointer to an error object that encapsulates the reason the document could not be created.

**Return Value**

The initialized `NSDocument` object, or, if the document could not be created, `nil`.

**Discussion**

The default implementation of this method just invokes `[self init]` and `[self setFileType:typeName]`.

You can override this method to perform initialization that must be done when creating new documents but should not be done when opening existing documents. Your override should typically invoke `super`, or at least it must invoke `init` (page 30), the `NSDocument` designated initializer, to initialize the `NSDocument` private instance variables.

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`NSDocument.h`

## isDocumentEdited

Returns `YES` if the receiver has changes that have not been saved, `NO` otherwise.

```
- (BOOL)isDocumentEdited
```

**Return Value**

`YES` if the receiver has been edited; otherwise, `NO`.

**Discussion**

The edited status of each document window reflects the document's edited status.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

`- updateChangeCount:` (page 58)

`- setDocumentEdited:` (NSWindow)

**Declared In**

`NSDocument.h`

## keepBackupFile

Returns whether the receiver should keep the backup files created before document data is written to a file (`NO` by default).

```
- (BOOL)keepBackupFile
```

**Return Value**
`NO` by default; subclasses can override to return `YES`, thereby causing backup files to be kept.

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

**See Also**
`- writeToFile:ofType:` (page 77)

**Declared In**
`NSDocument.h`

## lastComponentOfFileName

Returns the document name in terms of the scripting name property (the name a script writer would use to specify the document in a script).

```
- (NSString *)lastComponentOfFileName
```

**Return Value**
The scripting name of the document.

**Discussion**
Note that this name may be different than the name returned by `fileName` (page 70) or used in methods such as `writeToFile:ofType:` (page 77).

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

**See Also**
`- displayName` (page 23)

**Declared In**
`NSDocumentScripting.h`

## makeWindowControllers

Subclasses may override this method to create the initial window controller(s) for the document.

```
- (void)makeWindowControllers
```

**Discussion**
The base class implementation creates an `NSWindowController` object with `windowNibName` (page 62) and with the document as the file's owner if `windowNibName` (page 62) returns a name. If you override this method to create your own window controllers, be sure to use `addWindowController:` (page 19) to add them to the document after creating them.

This method is called by the `NSDocumentController` `open...` methods, but you might want to call it directly in some circumstances.

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

**See Also**
`- windowControllers` (page 60)

**Related Sample Code**
QTAudioExtractionPanel
QTKitAdvancedDocument
QTKitImport
QTKitPlayer
Sketch-112

**Declared In**
`NSDocument.h`

## objectSpecifier

Returns an object specifier for the document.

```
- (NSScriptObjectSpecifier *)objectSpecifier
```

**Return Value**
The document object specifier.

**Discussion**
An object specifier represents an AppleScript reference form, which is a natural-language expression such as `words 10 through 20` or `front document`. During script processing, an object contained by a document (such as the `second paragraph` or the `third rectangle`) may need to specify its container (the document).

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

**Declared In**
`NSDocumentScripting.h`

## preparePageLayout:

Invoked by `runModalPageLayoutWithPrintInfo:` (page 75) and `runModalPageLayoutWithPrintInfo:delegate:didRunSelector:contextInfo:` (page 44) to do any customization of the Page Layout panel *`pageLayout`*, such as adding an accessory view.

```
- (BOOL)preparePageLayout:(NSPageLayout *)pageLayout
```

**Parameters**

*`pageLayout`*
The page layout panel to prepare.

**Return Value**
`YES` if successfully prepared; otherwise, `NO`.

**Discussion**
The default implementation is empty and returns `YES`.

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

**Declared In**
`NSDocument.h`

## prepareSavePanel:

Invoked by `runModalSavePanelForSaveOperation:delegate:didSaveSelector:contextInfo:` (page 45) to do any customization of the given Save panel.

```
- (BOOL)prepareSavePanel:(NSSavePanel *)savePanel
```

**Parameters**

*savePanel*
The Save panel.

**Return Value**
`YES` if successfully prepared; otherwise, `NO`.

**Discussion**
The default implementation is empty and returns `YES`.

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

**Declared In**
`NSDocument.h`

## presentError:

Presents an error alert to the user as a modal panel.

```
- (BOOL)presentError:(NSError *)error
```

**Parameters**

*error*
The error object encapsulating the information to present to the user.

**Return Value**
`YES` if error recovery was done; otherwise, `NO`.

**Discussion**
This method does not return until the user dismisses the alert and, if the error has recovery options and a recovery delegate, the error's recovery delegate is sent an `attemptRecoveryFromError:optionIndex:` message.

The `NSDocument` default implementation of this method is equivalent to that of `NSResponder` and treats the shared `NSDocumentController` as the next responder and forwards these messages to it.

The default implementation of this method invokes `willPresentError:` (page 59) to give subclasses an opportunity to customize error presentation. You should not override this method but should instead override `willPresentError:` (page 59).

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

**See Also**
– `willPresentError:` (page 59)
– `presentError:modalForWindow:delegate:didPresentSelector:contextInfo:` (page 37)

**Declared In**
`NSDocument.h`

## presentError:modalForWindow:delegate:didPresentSelector:contextInfo:

Presents an error alert to the user as a modal panel.

```
- (void)presentError:(NSError *)error modalForWindow:(NSWindow *)window
    delegate:(id)delegate didPresentSelector:(SEL)didPresentSelector
    contextInfo:(void *)contextInfo
```

**Parameters**

*error*
The error object encapsulating the information to present to the user.

*window*
The window to which the modal alert belongs.

*delegate*
The delegate to which the selector message is sent.

*didPresentSelector*
The selector of the message sent to the delegate.

*contextInfo*
Object passed with the callback to provide any additional context information.

**Discussion**
When the user dismisses the alert and any recovery possible for the error and chosen by the user has been attempted, sends the message *didPresentSelector* to the specified *delegate*. The method selected by *didPresentSelector* must have the same signature as:

```
- (void)didPresentErrorWithRecovery:(BOOL)didRecover contextInfo:(void
*)contextInfo
```

The `NSDocument` default implementation of this method is equivalent to that of `NSResponder` and treats the shared `NSDocumentController` object as the next responder and forwards these messages to it. The default implementations of several `NSDocument` methods invoke this method.

The default implementation of this method invokes `willPresentError:` (page 59) to give subclasses an opportunity to customize error presentation. You should not override this method but should instead override `willPresentError:`.

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

**See Also**

`- presentError:` (page 36)

`- willPresentError:` (page 59)

**Declared In**

`NSDocument.h`

## printDocument:

Prints the receiver in response to the user choosing the Print menu command.

```
- (IBAction)printDocument:(id)sender
```

**Parameters**

*sender*

The control sending the message.

**Discussion**

An `NSDocument` object receives this action message as it travels up the responder chain. The default implementation invokes `printDocumentWithSettings:showPrintPanel:delegate:didPrintSelector:contextInfo:` (page 38).

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

`- printInfo` (page 39)

`- runPageLayout:` (page 46)

`- setPrintInfo:` (page 53)

`- shouldChangePrintInfo:` (page 55)

**Declared In**

`NSDocument.h`

## printDocumentWithSettings:showPrintPanel:delegate:didPrintSelector:contextInfo:

Prints the document.

```
- (void)printDocumentWithSettings:(NSDictionary *)printSettings
    showPrintPanel:(BOOL)showPrintPanel delegate:(id)delegate
    didPrintSelector:(SEL)didPrintSelector contextInfo:(void *)contextInfo
```

**Parameters**

*printSettings*

The print settings dictionary to use.

*showPrintPanel*

A Boolean value indicating whether the print panel is shown.

*delegate*

The delegate to which the selector message is sent.

*didPrintSelector*
: The selector of the message sent to the delegate.

*contextInfo*
: Object passed with the callback to provide any additional context information.

**Discussion**
If showing of the print panel is specified by *showPrintPanel*, the method presents it first and prints only if the user approves the panel. The `NSPrintInfo` attributes in the passed-in *printSettings* dictionary are added to a copy of the document's print info, and the resulting print info settings are used for the operation. When printing is complete or canceled, the method sends the message selected by *didPrintSelector* to the *delegate*, with the *contextInfo* as the last argument. The method selected by *didPrintSelector* must have the same signature as:

```
- (void)document:(NSDocument *)document didPrint:(BOOL)didPrintSuccessfully
contextInfo: (void *)contextInfo
```

The default implementation of this method invokes `printOperationWithSettings:error:` (page 40). If `nil` is returned it presents the error to the user in a document-modal panel before messaging the delegate. Otherwise it invokes `[thePrintOperation setShowsPrintPanel:showPrintPanel]` then `[self runModalPrintOperation:thePrintOperation delegate:delegate didRunSelector:didPrintSelector contextInfo:contextInfo]`.

For backward binary compatibility with Mac OS X v10.3 and earlier, the default implementation of this method invokes `printShowingPrintPanel:` (page 73) if it is overridden. When doing this it uses private functionality to arrange for the print settings to take effect (despite the fact that the override of `printShowingPrintPanel:` can't possibly know about them) and to get notified when the print operation has been completed, so it can message the delegate at the correct time. Correct messaging of the delegate is necessary for correct handling of the Print Apple event.

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

**See Also**
`- printOperationWithSettings:error:` (page 40)

**Declared In**
`NSDocument.h`

## printInfo

Returns the receiver's customized `NSPrintInfo` object or the default `NSPrintInfo` instance.

```
- (NSPrintInfo *)printInfo
```

**Return Value**
The receiver's `NSPrintInfo` object.

**Discussion**
The document's copy of the `NSPrintInfo` object can either be directly set or set as a result of running the Page Layout panel. A subclass can override this method to always return the shared `NSPrintInfo` instance if it does not want its own copy.

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

**See Also**
- `runPageLayout:` (page 46)
- `setPrintInfo:` (page 53)
- `shouldChangePrintInfo:` (page 55)

**Related Sample Code**
Sketch-112

**Declared In**
`NSDocument.h`

## printOperationWithSettings:error:

Creates a print operation and returns it if successful.

```
- (NSPrintOperation *)printOperationWithSettings:(NSDictionary *)printSettings
    error:(NSError **)outError
```

**Parameters**

*`printSettings`*
The print settings dictionary to use.

*`outError`*
On return, If the print operation could not be created, a pointer to an error object that encapsulates the reason it could not be created.

**Return Value**
The print operation, or `nil` if unsuccessful.

**Discussion**
The print operation can be run to print the document's current contents. The `NSPrintInfo` attributes in the passed-in *`printSettings`* dictionary are added to a copy of the document's print info, and the resulting print info is used for the operation. The default implementation of this method does nothing. You must override it to enable printing in your application.

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

**See Also**
- `printDocumentWithSettings:showPrintPanel:delegate:didPrintSelector:contextInfo:` (page 38)

**Declared In**
`NSDocument.h`

## readFromData:ofType:error:

Sets the contents of this document by reading from data of a specified type and returns `YES` if successful.

```
- (BOOL)readFromData:(NSData *)data ofType:(NSString *)typeName error:(NSError
    **)outError
```

**Parameters**

*`data`*
: The data object from which the document contents are read.

*`typeName`*
: The string that identifies the document type.

*`outError`*
: On return, If the document contents could not be read, a pointer to an error object that encapsulates the reason they could not be read.

**Return Value**

`YES` if the document contents could be read; otherwise, `NO`.

**Discussion**

The default implementation of this method throws an exception because at least one of the three reading methods (this method, `readFromURL:ofType:error:` (page 42), `readFromFileWrapper:ofType:error:` (page 41)), or every method that may invoke `readFromURL:ofType:error:` (page 42), must be overridden.

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`NSDocument.h`

## readFromFileWrapper:ofType:error:

Sets the contents of this document by reading from a file wrapper of a specified type.

```
- (BOOL)readFromFileWrapper:(NSFileWrapper *)fileWrapper ofType:(NSString *)typeName
     error:(NSError **)outError
```

**Parameters**

*`fileWrapper`*
: The file wrapper from which the document contents are read.

*`typeName`*
: The string that identifies the document type.

*`outError`*
: On return, If the document contents could not be read, a pointer to an error object that encapsulates the reason they could not be read.

**Return Value**

`YES` if the document contents could be read; otherwise, `NO`.

**Discussion**

The default implementation of this method invokes `[self readFromData:[fileWrapper regularFileContents] ofType:typeName error:outError]`.

For backward binary compatibility with Mac OS X v10.3 and earlier, the default implementation of this method instead invokes `[self loadFileWrapperRepresentation:fileWrapper ofType:typeName]` if `loadFileWrapperRepresentation:ofType:` (page 73) is overridden.

**Availability**

Available in Mac OS X v10.4 and later.

**See Also**

`- readFromURL:ofType:error:` (page 42)

`- readFromData:ofType:error:` (page 40)

**Declared In**

`NSDocument.h`

## readFromURL:ofType:error:

Sets the contents of this document by reading from a file or file package, of a specified type, located by a URL.

```
- (BOOL)readFromURL:(NSURL *)absoluteURL ofType:(NSString *)typeName error:(NSError
    **)outError
```

**Parameters**

*absoluteURL*

The location from which the document contents are read.

*typeName*

The string that identifies the document type.

*outError*

On return, If the document contents could not be read, a pointer to an error object that encapsulates the reason they could not be read.

**Return Value**

`YES` if the document contents could be read; otherwise, `NO`.

**Discussion**

The default implementation of this method just creates an NSFileWrapper and invokes `[self readFromFileWrapper:theFileWrapper ofType:typeName error:outError]`.

For backward binary compatibility with Mac OS X v10.3 and earlier, the default implementation of this method instead invokes `[self readFromFile:[absoluteURL path] ofType:typeName]` if `readFromFile:ofType:` (page 74) is overridden and the URL uses the `file:` scheme.

**Availability**

Available in Mac OS X v10.4 and later.

**See Also**

`- readFromFileWrapper:ofType:error:` (page 41)

`- readFromData:ofType:error:` (page 40)

**Declared In**

`NSDocument.h`

## removeWindowController:

Removes the specified window controller from the receiver's array of window controllers.

```
- (void)removeWindowController:(NSWindowController *)windowController
```

**Parameters**

*windowController*

The window controller that is removed.

**Discussion**

A document with no window controllers is not necessarily closed. However, a window controller can be set to close its associated document when the window is closed or the window controller is deallocated.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

`- shouldCloseDocument` (`NSWindowController`)

**Declared In**

`NSDocument.h`

## revertDocumentToSaved:

The action of the File menu item Revert in a document-based application.

```
- (IBAction)revertDocumentToSaved:(id)sender
```

**Parameters**

*sender*

The control sending the message.

**Discussion**

The default implementation of this method presents an alert dialog giving the user the opportunity to cancel the operation. If the user chooses to continue, the method ensures that any editor registered using the Cocoa Bindings `NSEditorRegistration` informal protocol has discarded its changes and then invokes `revertToContentsOfURL:ofType:error:` (page 43). If that returns `NO`, the method presents the error to the user in an document-modal alert dialog.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

`- updateChangeCount:` (page 58)

**Declared In**

`NSDocument.h`

## revertToContentsOfURL:ofType:error:

Discards all unsaved document modifications and replaces the document's contents by reading a file or file package located by a URL of a specified type.

```
- (BOOL)revertToContentsOfURL:(NSURL *)absoluteURL ofType:(NSString *)typeName
    error:(NSError **)outError
```

**Parameters**

*absoluteURL*
: The location from which the document contents are read.

*typeName*
: The string that identifies the document type.

*outError*
: On return, If the document could not be reverted, a pointer to an error object that encapsulates the reason it could not be reverted.

**Return Value**

`YES` if the document could be reverted; otherwise, `NO`.

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`NSDocument.h`

## runModalPageLayoutWithPrintInfo:delegate:didRunSelector:contextInfo:

Runs the modal page layout panel with the receiver's printing information object

```
- (void)runModalPageLayoutWithPrintInfo:(NSPrintInfo *)printInfo
    delegate:(id)delegate didRunSelector:(SEL)didRunSelector contextInfo:(void
    *)contextInfo
```

**Parameters**

*printInfo*
: The `NSPrintInfo` object for the page layout panel to use.

*delegate*
: The delegate to which the selector message is sent.

*didRunSelector*
: The selector of the message sent to the delegate.

*contextInfo*
: Object passed with the callback to provide any additional context information.

**Discussion**

Invoked from the action method `runPageLayout:` (page 46). Presents the page layout panel application modally if there is no document window to which it can be presented document modally.

When the panel is dismissed, *delegate* is sent a *didRunSelector* message. The *didRunSelector* callback method should have the following signature:

```
- (void)documentDidRunModalPageLayout:(NSDocument *)document
accepted:(BOOL)accepted  contextInfo:(void *)contextInfo
```

**Availability**

Available in Mac OS X v10.0 and later.

**Declared In**

`NSDocument.h`

## runModalPrintOperation:delegate:didRunSelector:contextInfo:

Runs the specified print operation modally.

```
- (void)runModalPrintOperation:(NSPrintOperation *)printOperation
    delegate:(id)delegate didRunSelector:(SEL)didRunSelector contextInfo:(void
    *)contextInfo
```

**Parameters**

*printOperation*
: The print operation to run.

*delegate*
: The delegate to which the selector message is sent.

*didRunSelector*
: The selector of the message sent to the delegate.

*contextInfo*
: Object passed with the callback to provide any additional context information.

**Discussion**
Overrides of `printShowingPrintPanel:` (page 73) can invoke this method.

When the panel is dismissed, *delegate* is sent a *didRunSelector* message. Pass the *contextInfo* object with the callback. The *didRunSelector* callback method should have the following signature:

```
- (void)documentDidRunModalPrintOperation:(NSDocument *)document
success:(BOOL)success  contextInfo:(void *)contextInfo
```

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

**Declared In**
`NSDocument.h`

## runModalSavePanelForSaveOperation:delegate:didSaveSelector:contextInfo:

Presents a modal Save panel to the user, then tries to save the document if the user approves the panel.

```
- (void)runModalSavePanelForSaveOperation:(NSSaveOperationType)saveOperation
    delegate:(id)delegate didSaveSelector:(SEL)didSaveSelector contextInfo:(void
    *)contextInfo
```

**Parameters**

*saveOperation*
: The type of save operation.

*delegate*
: The delegate to which the selector message is sent.

*didSaveSelector*
: The selector of the message sent to the delegate.

*contextInfo*
: Object passed with the callback to provide any additional context information.

**Discussion**
When saving is completed, regardless of success or failure, or has been canceled, sends the message selected by *didSaveSelector* to the *delegate*, with *contextInfo* as the last argument. The method selected by *didSaveSelector* must have the same signature as:

```
- (void)document:(NSDocument *)doc didSave:(BOOL)didSave contextInfo:(void
*)contextInfo
```

Invoked from `saveDocumentWithDelegate:didSaveSelector:contextInfo:` (page 48), and the action methods `saveDocumentAs:` (page 47) and `saveDocumentTo:` (page 48). The default implementation of this method first makes sure that any editor registered using the Cocoa Bindings `NSEditorRegistration` informal protocol has committed its changes, then creates a Save panel, adds a standard file format accessory view (if there is more than one file type for the user to choose from and `shouldRunSavePanelWithAccessoryView` (page 56) returns `YES`), sets various attributes of the panel, invokes `prepareSavePanel:` (page 36) to provide an opportunity for customization, then presents the panel. If the user approves the panel, the default implementation sends the message `saveToURL:ofType:forSaveOperation:delegate:didSaveSelector:contextInfo:` (page 49).

For backward binary compatibility with Mac OS 10.3 and earlier, the default implementation of this method instead invokes the deprecated `saveToFile:saveOperation:delegate:didSaveSelector:contextInfo:` (page 76) if it is overridden, even if the user cancels the panel.

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

**Declared In**
`NSDocument.h`

## runPageLayout:

The action method invoked in the receiver as first responder when the user chooses the Page Setup menu command.

```
- (IBAction)runPageLayout:(id)sender
```

**Parameters**

*sender*
The control sending the message.

**Discussion**
The default implementation invokes `runModalPageLayoutWithPrintInfo:delegate:didRunSelector:contextInfo:` (page 44) with the document's current NSPrintInfo object as argument; if the user clicks the OK button and the document authorizes changes to its printing information (`shouldChangePrintInfo:` (page 55)), the method sets the document's new `NSPrintInfo` object and increments the document's change count.

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

**See Also**
`– setPrintInfo:` (page 53)
`– updateChangeCount:` (page 58)

**Declared In**
`NSDocument.h`

## saveDocument:

The action method invoked in the receiver as first responder when the user chooses the Save menu command.

```
- (IBAction)saveDocument:(id)sender
```

**Parameters**

*sender*
The control sending the message.

**Discussion**
The default implementation saves the document in two different ways, depending on whether the document has a file path and a document type assigned. If path and type are assigned, it simply writes the document under its current file path and type after making a backup copy of the previous file. If the document is new (no file path and type), it runs the modal Save panel to get the file location under which to save the document. It writes the document to this file, sets the document's file location and document type (if a native type), and clears the document's edited status.

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

**See Also**
`- saveDocumentWithDelegate:didSaveSelector:contextInfo:` (page 48)
`- setFileName:` (page 77)
`- setFileType:` (page 51)
`- updateChangeCount:` (page 58)

**Declared In**
`NSDocument.h`

## saveDocumentAs:

The action method invoked in the receiver as first responder when the user chooses the Save As menu command.

```
- (IBAction)saveDocumentAs:(id)sender
```

**Parameters**

*sender*
The control sending the message.

**Discussion**
The default implementation runs the modal Save panel to get the file location under which to save the document. It writes the document to this file, sets the document's file location and document type (if a native type), and clears the document's edited status.

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

**See Also**

- `saveDocumentWithDelegate:didSaveSelector:contextInfo:` (page 48)
- `setFileName:` (page 77)
- `setFileType:` (page 51)
- `updateChangeCount:` (page 58)

**Declared In**

`NSDocument.h`

## saveDocumentTo:

The action method invoked in the receiver as first responder when the user chooses the Save To menu command.

```
- (IBAction)saveDocumentTo:(id)sender
```

**Parameters**

*sender*
: The control sending the message.

**Discussion**

The default implementation is identical to `saveDocumentAs:` except that this method doesn't clear the document's edited status and doesn't reset file location and document type if the document is a native type.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

- `saveDocumentWithDelegate:didSaveSelector:contextInfo:` (page 48)

**Declared In**

`NSDocument.h`

## saveDocumentWithDelegate:didSaveSelector:contextInfo:

Saves the document.

```
- (void)saveDocumentWithDelegate:(id)delegate didSaveSelector:(SEL)didSaveSelector
    contextInfo:(void *)contextInfo
```

**Parameters**

*delegate*
: The delegate to which the selector message is sent.

*didSaveSelector*
: The selector of the message sent to the delegate.

*contextInfo*
: Object passed with the callback to provide any additional context information.

**Discussion**

If an `NSSaveOperation` can be performed without further user intervention (at the very least, neither `fileURL` (page 27) nor `fileType` (page 26) return `nil`), then the method immediately saves the document. Otherwise, it presents a Save panel to the user and saves the document if the user approves the panel. When saving has been completed or canceled, the method sends the message selected by *`didSaveSelector`* to the *`delegate`*, with the *`contextInfo`* as the last argument.

As of Mac OS X v10.5, this method checks to see if the document's file has been modified since the document was opened or most recently saved or reverted, in addition to the checking for file moving, renaming, and trashing that it has done since Mac OS X v10.1. When it senses file modification it presents an alert telling the user "This document's file has been changed by another application since you opened or saved it," giving them the choice of saving or not saving. For backward binary compatibility this is only done in applications linked against Mac OS X v10.5 or later.

The *`didSaveSelector`* callback method should have the following signature:

```
- (void)document:(NSDocument *)doc didSave:(BOOL)didSave contextInfo:(void
*)contextInfo
```

**Availability**

Available in Mac OS X v10.0 and later.

**Declared In**

`NSDocument.h`

## saveToURL:ofType:forSaveOperation:delegate:didSaveSelector:contextInfo:

Saves the contents of the document to a file or file package located by a URL, formatted to a specified type, for a particular kind of save operation.

```
- (void)saveToURL:(NSURL *)absoluteURL ofType:(NSString *)typeName
    forSaveOperation:(NSSaveOperationType)saveOperation delegate:(id)delegate
    didSaveSelector:(SEL)didSaveSelector contextInfo:(void *)contextInfo
```

**Parameters**

*`absoluteURL`*
: The location of the file or file package to which the document contents are saved.

*`typeName`*
: The string that identifies the document type.

*`saveOperation`*
: The type of save operation.

*`delegate`*
: The delegate to which the selector message is sent.

*`didSaveSelector`*
: The selector of the message sent to the delegate.

*`contextInfo`*
: Object passed with the callback to provide any additional context information.

**Discussion**

When saving is completed, regardless of success or failure, the method sends the message selected by *`didSaveSelector`* to the *`delegate`*, with the *`contextInfo`* as the last argument. The method selected by *`didSaveSelector`* must have the same signature as:

```
- (void)document:(NSDocument *)document didSave:(BOOL)didSaveSuccessfully
contextInfo:(void  *)contextInfo;
```

The default implementation of this method invokes `[self saveToURL:absoluteURL ofType:typeName forSaveOperation:saveOperation error:&anError]` and, if `NO` is returned, presents the error to the user in a document-modal panel before messaging the delegate.

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

**See Also**
`– saveToURL:ofType:forSaveOperation:error:` (page 50)

**Declared In**
`NSDocument.h`

## saveToURL:ofType:forSaveOperation:error:

Saves the contents of the document to a file or file package located by a URL, formatted to a specified type, for a particular kind of save operation, and returns `YES` if successful.

```
- (BOOL)saveToURL:(NSURL *)absoluteURL ofType:(NSString *)typeName
    forSaveOperation:(NSSaveOperationType)saveOperation error:(NSError **)outError
```

**Parameters**

*absoluteURL*
The location of the file or file package to which the document contents are saved.

*typeName*
The string that identifies the document type.

*saveOperation*
The type of save operation.

*outError*
On return, If the document contents could not be saved, a pointer to an error object that encapsulates the reason they could not be saved.

**Return Value**
`YES` if the document contents were successfully saved; otherwise, `NO`.

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

**See Also**
`– saveToURL:ofType:forSaveOperation:delegate:didSaveSelector:contextInfo:` (page 49)

**Declared In**
`NSDocument.h`

## setAutosavedContentsFileURL:

Sets the location of the most recently autosaved document contents.

```
- (void)setAutosavedContentsFileURL:(NSURL *)absoluteURL
```

**Parameters**

*absoluteURL*

The location of the most recently autosaved document contents.

**Discussion**

The default implementation of this method records the URL and notifies the shared document controller that this document should be automatically reopened if the application quits or crashes before the document is saved.

**Availability**

Available in Mac OS X v10.4 and later.

**See Also**

`- autosavedContentsFileURL` (page 19)

**Declared In**

`NSDocument.h`

## setFileModificationDate:

Sets the last known modification date of the document's on-disk representation to the given modification date.

```
- (void)setFileModificationDate:(NSDate *)modificationDate
```

**Parameters**

*modificationDate*

The date to which the file modification date is set.

**Discussion**

The `NSDocument` default file saving machinery uses this information to warn the user when the on-disk representation of an open document has been modified by something other than the current application.

**Availability**

Available in Mac OS X v10.4 and later.

**See Also**

`- fileModificationDate` (page 25)

**Declared In**

`NSDocument.h`

## setFileType:

Sets the document type under which the file is saved.

```
- (void)setFileType:(NSString *)docType
```

**Parameters**

*docType*

The string that identifies the document type.

**Discussion**
The document type affects how the data is filtered when it is written to or read from a file. This method isn't for changing the document's format; it's just for initially recording the document's format during opening or saving.

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

**See Also**
`- fileType` (page 26)

**Declared In**
`NSDocument.h`

## setFileURL:

Sets the location of the document's on-disk representation.

```
- (void)setFileURL:(NSURL *)absoluteURL
```

**Parameters**

*`absoluteURL`*
The document's location.

**Discussion**
This method doesn't actually rename the document; it's just for recording the document's location during initial opening or saving. The default implementation of this method just records the URL so that the default implementation of `fileURL` (page 27) can return it.

For backward binary compatibility with Mac OS X v10.3 and earlier, if `setFileName:` (page 77) is overridden and the URL is `nil` or uses the `file:` scheme, the default implementation of this method instead invokes `[self setFileName:[absoluteURL path]]`.

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

**See Also**
`- fileURL` (page 27)

**Related Sample Code**
QTMetadataEditor

**Declared In**
`NSDocument.h`

## setHasUndoManager:

Sets whether the receiver has its own `NSUndoManager` object.

```
- (void)setHasUndoManager:(BOOL)flag
```

**Parameters**

*flag*

A Boolean value setting whether the receiver should own an `NSUndoManager` object.

**Discussion**

If *flag* is `NO` and the receiver currently owns an `NSUndoManager` object, the `NSUndoManager` object is released after being removed as an observer of undo-related notifications.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

`– hasUndoManager` (page 30)

**Declared In**

`NSDocument.h`

## setLastComponentOfFileName:

Sets the document name to the given string in terms of the scripting name property (the name a script writer would use to specify the document in a script).

```
- (void)setLastComponentOfFileName:(NSString *)str
```

**Parameters**

*str*

The scripting name of the document.

**Discussion**

Note that this name may be different than the name used in `setFileName:` (page 77).

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

`– displayName` (page 23)

**Declared In**

`NSDocumentScripting.h`

## setPrintInfo:

Sets the receiver's `NSPrintInfo` object.

```
- (void)setPrintInfo:(NSPrintInfo *)printInfo
```

**Parameters**

*printInfo*

The `NSPrintInfo` object for the receiver to use.

**Discussion**

This `NSPrintInfo` object is used in laying out the document for printing.

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

**See Also**
`- printInfo` (page 39)

**Declared In**
`NSDocument.h`

## setUndoManager:

Sets the undo manager owned by the receiver to the specified undo manager and releases any undo manager currently owned by the receiver.

```
- (void)setUndoManager:(NSUndoManager *)undoManager
```

**Parameters**

*undoManager*
: The undo manager to be owned by the receiver; may be `nil`.

**Discussion**
If *undoManager* is `nil`, it turns off the `hasUndoManager` flag. If *undoManager* is non-`nil`, it adds the receiver as an observer of `NSUndoManagerDidUndoChangeNotification`, `NSUndoManagerDidRedoChangeNotification`, and `NSUndoManagerWillCloseUndoGroupNotification`.

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

**See Also**
`- undoManager` (page 57)

**Declared In**
`NSDocument.h`

## setWindow:

Sets the `window` Interface Builder outlet of this class.

```
- (void)setWindow:(NSWindow *)aWindow
```

**Parameters**

*aWindow*
: The window to which the receiver's `window` outlet points.

**Discussion**
This method is invoked automatically during the loading of any nib for which this document is the file's owner, if the file's owner `window` outlet is connected in the nib. You should not invoke this method directly, and typically you would not override it either.

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

**Declared In**
`NSDocument.h`

## shouldChangePrintInfo:

Returns a Boolean value indicating whether the receiver should allow changes to the default `NSPrintInfo` object used in printing the document.

```
- (BOOL)shouldChangePrintInfo:(NSPrintInfo *)newPrintInfo
```

**Parameters**

*`newPrintInfo`*
The `NSPrintInfo` object that is the result of the user approving the page layout panel presented by `runPageLayout:` (page 46).

**Return Value**
`YES` by default; subclasses can override this method to return `NO`.

**Discussion**
This method is invoked by the `runPageLayout:` (page 46) method, which sets a new `NSPrintInfo`object for the document only if this method returns `YES`.

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

**Declared In**
`NSDocument.h`

## shouldCloseWindowController:

Gives the user an opportunity to save the document. (**Deprecated.** Use `shouldCloseWindowController:delegate:shouldCloseSelector:contextInfo:` (page 56) instead.)

```
- (BOOL)shouldCloseWindowController:(NSWindowController *)windowController
```

**Discussion**
If closing the *`windowController`* would cause the receiver to be closed, invokes `canCloseDocumentWithDelegate:shouldCloseSelector:contextInfo:` (page 21) to display a Save panel and give the user an opportunity to save the document. Returns the return value of `canCloseDocumentWithDelegate:shouldCloseSelector:contextInfo:`. Note that the receiver doesn't close until its window controller closes.

**Availability**
Available in Mac OS X v10.0 through Mac OS X v10.3.

**Declared In**
`NSDocument.h`

### shouldCloseWindowController:delegate:shouldCloseSelector:contextInfo:

Invokes *shouldCloseSelector* with the result of canCloseDocumentWithDelegate:shouldCloseSelector:contextInfo: (page 21) if the the specified window controller that is closing is the last one or is marked as causing the document to close.

```
- (void)shouldCloseWindowController:(NSWindowController *)windowController
    delegate:(id)delegate shouldCloseSelector:(SEL)shouldCloseSelector
    contextInfo:(void *)contextInfo
```

**Parameters**

*windowController*
: The window controller that is closed.

*delegate*
: The delegate to which the selector message is sent.

*shouldCloseSelector*
: The selector of the message sent to the delegate.

*contextInfo*
: Object passed with the callback to provide any additional context information.

**Discussion**
Otherwise it invokes *shouldCloseSelector* with YES. This method is called automatically by NSWindow for any window that has a window controller and a document associated with it. NSWindow calls this method prior to sending its *delegate* the windowShouldClose: message. Pass the *contextInfo* object with the callback.

The *shouldCloseSelector* callback method should have the following signature:

```
- (void)document:(NSDocument *)document shouldClose:(BOOL)shouldClose
contextInfo:(void  *)contextInfo
```

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

**Declared In**
NSDocument.h

### shouldRunSavePanelWithAccessoryView

Returns YES by default; as a result, when NSDocument displays the Save panel, it includes an accessory view containing a pop-up menu of supported writable document types.

```
- (BOOL)shouldRunSavePanelWithAccessoryView
```

**Return Value**
YES by default; subclasses can override to return NO, thereby excluding the accessory view from the Save panel.

**Discussion**
Here is an example implementation:

```
- (BOOL)shouldRunSavePanelWithAccessoryView {
    return [self fileName] == nil;
}
```

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

**See Also**
`- runModalSavePanelForSaveOperation:delegate:didSaveSelector:contextInfo:` (page 45)

**Declared In**
`NSDocument.h`

## showWindows

Displays all of the document's windows, bringing them to the front and making them main or key as necessary.

```
- (void)showWindows
```

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

**Related Sample Code**
QTAudioExtractionPanel
QTKitAdvancedDocument
QTKitImport
QTKitPlayer
QTMetadataEditor

**Declared In**
`NSDocument.h`

## undoManager

Returns the receiver's undo manager.

```
- (NSUndoManager *)undoManager
```

**Return Value**
The `NSUndoManager` object used by the receiver or `nil` if the receiver should not own one.

**Discussion**
If the undo manager doesn't exist and `hasUndoManager` returns `YES`, the method creates one and invokes `setUndoManager:` with the `NSUndoManager` as argument.

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

**Related Sample Code**
File Wrappers with Core Data Documents
Sketch-112
Squiggles

**Declared In**
`NSDocument.h`

## updateChangeCount:

Updates the receiver's change count according to the given change type.

```
- (void)updateChangeCount:(NSDocumentChangeType)changeType
```

**Parameters**

*changeType*

The type of change made to the document.

**Discussion**

The change count indicates the document's edited status; if the change count is 0, the document has no changes to save, and if the change count is greater than 0, the document has been edited and is unsaved. The `changeType` is described in "Constants" (page 66). If you are implementing undo and redo in an application, you should increment the change count every time you create an undo group and decrement the change count when an undo or redo operation is performed.

Note that if you are using the `NSDocument` default undo/redo features, setting the document's edited status by updating the change count happens automatically. You only need to invoke this method when you are not using these features.

**Availability**

Available in Mac OS X v10.0 and later.

**Related Sample Code**

QTAudioExtractionPanel

QTKitImport

QTKitPlayer

**Declared In**

`NSDocument.h`

## validateMenuItem:

Validates the Revert menu item and items selected from the Save panel's pop-up list of writable document types items. (**Deprecated.** Use `validateUserInterfaceItem:` (page 59) instead.)

```
- (BOOL)validateMenuItem:(NSMenuItem *)anItem
```

**Discussion**

Returns `YES` if *anItem* should be enabled, `NO` otherwise. Returns `YES` for Revert if the document has been edited and a file exists for the document. Returns `YES` for an item representing a writable type if, during a Save or Save As operation, it is a native type for the document. Subclasses can override this method to perform additional validations.

**Availability**

Available in Mac OS X v10.0 through Mac OS X v10.3.

**Declared In**

`NSDocument.h`

## validateUserInterfaceItem:

Validates the specified user interface item that the receiver manages.

```
- (BOOL)validateUserInterfaceItem:(id < NSValidatedUserInterfaceItem >)anItem
```

**Parameters**

*anItem*

The user interface item to validate.

**Return Value**

`YES` if the item is valid; otherwise, `NO`.

**Discussion**

These items currently include only Revert (which is enabled only if the document has a `fileName` (page 70)) and Save. You can override this method to add more selectors validated by your document subclass.

**Availability**

Available in Mac OS X v10.0 and later.

**Declared In**

`NSDocument.h`

## willPresentError:

Called when the receiver is about to present an error.

```
- (NSError *)willPresentError:(NSError *)error
```

**Parameters**

*error*

The error object that is about to be presented to the user.

**Return Value**

The error that should actually be presented.

**Discussion**

The default implementation of this method merely returns the passed-in error. The returned error may simply be forwarded to the document controller.

You can override this method to customize the presentation of errors by examining the passed-in error and, for example, returning more specific information. When you override this method always check the `NSError` object's domain and code to discriminate between errors whose presentation you want to customize and those you don't. For errors you don't want to customize, call the superclass implementation, passing the original error.

**Availability**

Available in Mac OS X v10.4 and later.

**See Also**

- `presentError:` (page 36)
- `presentError:modalForWindow:delegate:didPresentSelector:contextInfo:` (page 37)

**Declared In**

`NSDocument.h`

## windowControllerDidLoadNib:

Sent after the specified window controller loads a nib file if the receiver is the nib file's owner.

```
- (void)windowControllerDidLoadNib:(NSWindowController *)windowController
```

**Parameters**

*`windowController`*

The window controller that loads the nib file.

**Discussion**

See the class description for NSWindowController for additional information about nib files and the file's owner object.

Typically an `NSDocument` subclass overrides `windowNibName` (page 62) or `makeWindowControllers` (page 34), but not both. If `windowNibName` is overridden, the default implementation of `makeWindowControllers` will load the named nib file, making the `NSDocument` object the nib file's owner. In that case, you can override `windowControllerDidLoadNib:` and do custom processing after the nib file is loaded.

The default implementation of this method does nothing.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

- `windowControllerWillLoadNib:` (page 61)
- `windowControllers` (page 60)

**Declared In**

`NSDocument.h`

## windowControllers

Returns the receiver's current window controllers.

```
- (NSArray *)windowControllers
```

**Return Value**

An array containing `NSWindowController` objects belonging to the current document. If there are no window controllers, returns an empty `NSArray` object.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

- `makeWindowControllers` (page 34)
- `windowControllerDidLoadNib:` (page 60)
- `windowControllerWillLoadNib:` (page 61)
- `windowNibName` (page 62)

**Related Sample Code**

Sketch-112

**Declared In**
`NSDocument.h`

## windowControllerWillLoadNib:

Sent before the specified window controller loads a nib file if the receiver is the nib file's owner.

```
- (void)windowControllerWillLoadNib:(NSWindowController *)windowController
```

**Parameters**

*windowController*
The window controller that loads the nib file.

**Discussion**
See the class description for NSWindowController for additional information about nib files and the file's owner object.

Typically an `NSDocument` subclass overrides `windowNibName` (page 62) or `makeWindowControllers` (page 34), but not both. If `windowNibName` is overridden, the default implementation of `makeWindowControllers` will load the named nib file, making the NSDocument the nib file's owner. In that case, you can override `windowControllerWillLoadNib:` and do custom processing before the nib file is loaded.

The default implementation of this method does nothing.

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

**See Also**
`– windowControllerDidLoadNib:` (page 60)
`– windowControllers` (page 60)

**Declared In**
`NSDocument.h`

## windowForSheet

Returns the most appropriate window, of the windows associated with the receiver, to use as the parent window of a document-modal sheet.

```
- (NSWindow *)windowForSheet
```

**Return Value**
The window to use as the parent window of the sheet.

**Discussion**
May return `nil`, in which case the sender should present an application-modal panel. The `NSDocument` implementation of this method returns the window of the first window controller, or `[NSApp mainWindow]` if there are no window controllers or if the first window controller has no window.

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

**Declared In**
`NSDocument.h`

## windowNibName

Overridden by subclasses to return the name of the document's sole nib file.

```
- (NSString *)windowNibName
```

**Return Value**
The name of the document nib file.

**Discussion**
Using this name, `NSDocument` creates and instantiates a default instance of `NSWindowController` to manage the window. If your document has multiple nib files, each with its own single window, or if the default `NSWindowController` instance is not adequate for your purposes, you should override `makeWindowControllers`.

The default implementation returns `nil`.

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

**See Also**
`- windowControllers` (page 60)

**Related Sample Code**
CocoaSpeechSynthesisExample
iSpend
QTKitButtonTester
Simple Bindings Adoption
WritableFileDemo

**Declared In**
`NSDocument.h`

## writableTypesForSaveOperation:

Returns the names of the types to which this document can be saved for a specified kind of save operation.

```
- (NSArray *)writableTypesForSaveOperation:(NSSaveOperationType)saveOperation
```

**Parameters**

*`saveOperation`*
The kind of save operation.

**Return Value**
An array of `NSString` objects representing the writable document types.

**Discussion**
The save operation type is represented by *`saveOperation`*. For every kind of save operation except `NSSaveToOperation`, the returned array must only include types for which the the application can play the Editor role. For `NSSaveToOperation` the returned array may include types for which the application can

only play the Viewer role, and other types that the application can merely export. The default implementation of this method returns `[[self class] writableTypes]` with, except during `NSSaveToOperation`, types for which `isNativeType:` (page 17) returns `NO` filtered out.

You can override this method to limit the set of writable types when the document currently contains data that is not representable in all types. For example, you can disallow saving to RTF files when the document contains an attachment and can only be saved properly to RTFD files.

You can invoke this method when creating a custom save panel accessory view to present easily the same set of types as `NSDocument` does in its standard file format popup menu.

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

**Declared In**
`NSDocument.h`

## writeSafelyToURL:ofType:forSaveOperation:error:

Writes the contents of the document to a file or file package located by a URL.

```
- (BOOL)writeSafelyToURL:(NSURL *)absoluteURL ofType:(NSString *)typeName
    forSaveOperation:(NSSaveOperationType)saveOperation error:(NSError **)outError
```

**Parameters**

*absoluteURL*
The location to which the document contents are written.

*typeName*
The string that identifies the document type.

*saveOperation*
The type of save operation.

*outError*
On return, If the document contents could not be written, a pointer to an error object that encapsulates the reason they could not be written.

**Return Value**
`YES` if the document contents could be written; otherwise, `NO`.

**Discussion**
The default implementation of this method invokes `writeToURL:ofType:forSaveOperation:originalContentsURL:error:` (page 65). It also invokes `fileAttributesToWriteToURL:ofType:forSaveOperation:originalContentsURL:error:` (page 24) and writes the returned attributes, if any, to the file. It may copy some attributes from the old on-disk revision of the document at the same time, if applicable.

This method is responsible for doing document writing in a way that minimizes the danger of leaving the disk to which writing is being done in an inconsistent state in the event of an application crash, system crash, hardware failure, power outage, and so on. If you override this method, be sure to invoke the superclass implementation.

For `NSSaveOperation`, the default implementation of this method invokes `keepBackupFile` (page 34) to determine whether or not the old on-disk revision of the document, if there was one, should be preserved after being renamed.

For backward binary compatibility with Mac OS X v10.3 and earlier, the default implementation of this method instead invokes `writeWithBackupToFile:ofType:saveOperation:` (page 78) if that method is is overridden and the URL uses the `file:` scheme. The save operation in this case is never `NSAutosaveOperation`; `NSSaveToOperation` is used instead.

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

**See Also**
- `writeToURL:ofType:forSaveOperation:originalContentsURL:error:` (page 65)
- `fileAttributesToWriteToURL:ofType:forSaveOperation:originalContentsURL:error:` (page 24)

**Declared In**
`NSDocument.h`

## writeToURL:ofType:error:

Writes the contents of the document to a file or file package located by a URL, formatted to a specified type.

```
- (BOOL)writeToURL:(NSURL *)absoluteURL ofType:(NSString *)typeName error:(NSError
    **)outError
```

**Parameters**

*absoluteURL*
  The location to which the document contents are written.

*typeName*
  The string that identifies the document type.

*outError*
  On return, If the document contents could not be written, a pointer to an error object that encapsulates the reason they could not be written.

**Return Value**
`YES` if the document contents could be written; otherwise, `NO`.

**Discussion**
The default implementation of this method just invokes `[self fileWrapperOfType:typeName error:outError]` and writes the returned file wrapper to disk.

For backward binary compatibility with Mac OS X v10.3 and earlier, the default implementation of this method instead invokes `[self writeToFile:[absoluteURL path] ofType:typeName]` if `writeToFile:ofType:` (page 77) is overridden and the URL uses the `file:` scheme.

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

**See Also**
- `fileWrapperOfType:error:` (page 28)
- `dataOfType:error:` (page 22)

**Declared In**
`NSDocument.h`

### writeToURL:ofType:forSaveOperation:originalContentsURL:error:

Writes the contents of the document to a file or file package located by a URL.

```
- (BOOL)writeToURL:(NSURL *)absoluteURL ofType:(NSString *)typeName
    forSaveOperation:(NSSaveOperationType)saveOperation originalContentsURL:(NSURL
     *)absoluteOriginalContentsURL error:(NSError **)outError
```

**Parameters**

*absoluteURL*
: The location to which the document contents are written.

*typeName*
: The string that identifies the document type.

*saveOperation*
: The type of save operation.

*absoluteOriginalContentsURL*
: The location of the previously saved copy of the document (if not `nil`).

*outError*
: On return, If the document contents could not be written, a pointer to an error object that encapsulates the reason they could not be written.

**Return Value**

`YES` if the document contents could be written; otherwise, `NO`.

**Discussion**

The default implementation of this method merely invokes `[self writeToURL:absoluteURL ofType:typeName error:outError]`. You can override this method instead of one of the three simple writing methods (`writeToURL:ofType:error:` (page 64),`fileWrapperOfType:error:` (page 28), and `dataOfType:error:` (page 22)) if your document writing machinery needs access to the on-disk representation of the document revision that is about to be overwritten. The value of *absoluteURL* is often not the same as `[self fileURL]`. Other times it is not the same as the URL for the final save destination. Likewise, *absoluteOriginalContentsURL* is often not the same value as `[self fileURL]`. If *absoluteOriginalContentsURL* is `nil`, either the document has never been saved or the user deleted the document file since it was opened.

For backward binary compatibility with Mac OS X v10.3 and earlier, if `writeToFile:ofType:originalFile:saveOperation:` (page 78) is overridden and both URLs use the `file:` scheme, the default implementation of this method instead invokes:

```
[self writeToFile:[absoluteURL path]
      ofType:typeName
      originalFile:[absoluteOriginalContentsURL path]
      saveOperation:aSaveOperation];
```

The save operation used in this case is never `NSAutosaveOperation`; `NSSaveToOperation` is used instead.

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`NSDocument.h`

## Constants

### NSSaveOperationType

The following constants specify types of save operations. These values are used with method parameters of type `NSSaveOperationType`. Depending on the method, those parameters can affect the title of the Save panel, as well as the files displayed.

```
enum {
    NSSaveOperation   = 0,
    NSSaveAsOperation = 1,
    NSSaveToOperation = 2
    NSAutosaveOperation = 3
};
typedef NSUInteger NSSaveOperationType;
```

**Constants**

`NSSaveOperation`

Specifies a Save operation, the overwriting of a document's file or file package with the document's current contents.

Available in Mac OS X v10.0 and later.

Declared in `NSDocument.h`.

`NSSaveAsOperation`

Specifies a Save As operation, the writing of a document's current contents to a new file or file package, and then making the just-written file or file package the document's current one.

Available in Mac OS X v10.0 and later.

Declared in `NSDocument.h`.

`NSSaveToOperation`

Specifies a Save To operation, the writing of a document's current contents to a new file or file package without changing the document's current one.

Available in Mac OS X v10.0 and later.

Declared in `NSDocument.h`.

`NSAutosaveOperation`

Specifies an autosave operation, writing a document's contents to a file or file package separate from the document's current one, without changing the document's current one.

Available in Mac OS X v10.4 and later.

Declared in `NSDocument.h`.

**Availability**

Available in Mac OS X v10.0 and later.

**Declared In**

`NSDocument.h`

### NSDocumentChangeType

Change counts indicate a document's edit status. These constants indicate how a document should operate on its change count and are passed to `updateChangeCount:` (page 58).

```
enum {
    NSChangeDone = 0,
    NSChangeUndone = 1,
    NSChangeCleared = 2,
    NSChangeReadOtherContents = 3,
    NSChangeAutosaved = 4,
    NSChangeRedone = 5
};
typedef NSUInteger NSDocumentChangeType;
```

**Constants**

`NSChangeDone`

Increment change count. The value to pass to `updateChangeCount:` (page 58) to indicate that a single change has been done. For example, the built-in undo support of `NSDocument` passes this value whenever a document receives an `NSUndoManagerWillCloseUndoGroupNotification` from its own undo manager.

Available in Mac OS X v10.0 and later.

Declared in `NSDocument.h`.

`NSChangeUndone`

Decrement change count. A single change has been undone. For example, the built-in undo support of `NSDocument` passes this value whenever a document receives an `NSUndoManagerDidUndoChangeNotification` from its own undo manager.

Available in Mac OS X v10.0 and later.

Declared in `NSDocument.h`.

`NSChangeCleared`

Set change count to 0. The document has been synchronized with its file or file package. For example, `saveToURL:ofType:forSaveOperation:error:` (page 50) passes this value for a successful `NSSaveOperation` (page 66) or `NSSaveAsOperation` (page 66). The `revertDocumentToSaved:` (page 43) method does too.

Available in Mac OS X v10.0 and later.

Declared in `NSDocument.h`.

`NSChangeReadOtherContents`

The document has been initialized with the contents of a file or file package other than the one whose location would be returned by `fileURL` (page 27), and therefore can't possibly be synchronized with its persistent representation. For example, `initForURL:withContentsOfURL:ofType:error:` (page 31) passes this value when the two passed-in URLs are not equal to indicate that an autosaved document is being reopened.

Available in Mac OS X v10.4 and later.

Declared in `NSDocument.h`.

`NSChangeAutosaved`

The document's contents have been autosaved. For example, `saveToURL:ofType:forSaveOperation:error:` (page 50) passes this value for a successful `NSAutosaveOperation` (page 66).

Available in Mac OS X v10.4 and later.

Declared in `NSDocument.h`.

`NSChangeRedone`

A single change has been redone. For example, the built-in undo support of `NSDocument` passes this value whenever a document receives an `NSUndoManagerDidRedoChangeNotification` from its own undo manager.

Available in Mac OS X v10.5 and later.

Declared in `NSDocument.h`.

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

**Declared In**
`NSDocument.h`

# Deprecated NSDocument Methods

A method identified as deprecated has been superseded and may become unsupported in the future.

## Deprecated in Mac OS X v10.4

### dataRepresentationOfType:

A primitive method overridden by subclasses to return a data object that represents the data of the receiver in a given type. (Deprecated in Mac OS X v10.4. Use `dataOfType:error:` (page 22) instead.)

```
- (NSData *)dataRepresentationOfType:(NSString *)aType
```

**Discussion**

A primitive method overridden by subclasses to return a data object that represents the data of the receiver in a given type (*aType*). The default implementation `raises` an `NSInternalInconsistencyException`. This method is invoked by the default implementation of `fileWrapperRepresentationOfType:`.

*aType* is the type name corresponding to the value of the `CFBundleTypeName` entry in the document type's `Info.plist` dictionary.

Here is a typical implementation:

```
//Document type name
NSString *MyDocumentType = @"Rich Text Format (RTF) document";

...

- (NSData *)dataRepresentationOfType:(NSString *)aType {
    NSAssert([aType isEqualToString:MyDocumentType], @"Unknown  type");
    return [textView RTFFromRange:NSMakeRange(0, [[textView textStorage]
length])];
}
```

**Availability**

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

**See Also**

`- loadDataRepresentation:ofType:` (page 72)

**Declared In**

`NSDocument.h`

## fileAttributesToWriteToFile:ofType:saveOperation:

Returns the file attributes that should be written to the named document file of the specified type. (Deprecated in Mac OS X v10.4. Use `fileAttributesToWriteToURL:ofType:forSaveOperation:originalContentsURL:error:` (page 24) instead.)

```
- (NSDictionary *)fileAttributesToWriteToFile:(NSString *)fullDocumentPath
    ofType:(NSString *)docType saveOperation:(NSSaveOperationType)saveOperationType
```

**Discussion**
Returns the file attributes that should be written to the named document file of the specified type *docType*, as part of a particular *saveOperationType*. The set of valid file attributes is a subset of those understood by the `NSFileManager` class.

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

Deprecated in Mac OS X v10.4.

**Declared In**
`NSDocument.h`

## fileName

Returns the filename (as a fully qualified path) under which the receiver has been saved. (Deprecated in Mac OS X v10.4. Use `fileURL` (page 27) instead.)

```
- (NSString *)fileName
```

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

Deprecated in Mac OS X v10.4.

**See Also**
`- setFileName:` (page 77)

**Related Sample Code**
QTKitAdvancedDocument

QTKitProgressTester

ThreadsExporter

ThreadsImporter

ThreadsImportMovie

**Declared In**
`NSDocument.h`

## fileWrapperRepresentationOfType:

Returns an NSFileWrapper object that represents the data of the receiver in a given type. (Deprecated in Mac OS X v10.4. Use `fileWrapperOfType:error:` (page 28) instead.)

```
- (NSFileWrapper *)fileWrapperRepresentationOfType:(NSString *)aType
```

**Discussion**
Returns an `NSFileWrapper` object that represents the data of the receiver in a given type (*aType*). This method invokes `dataRepresentationOfType:` to get the data object from which to create a plain-file file wrapper. Subclasses can override this method if `dataRepresentationOfType:` is not adequate for their needs. This method is invoked by the default implementation of `writeToFile:ofType:` (page 77).

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

Deprecated in Mac OS X v10.4.

**See Also**
`- loadFileWrapperRepresentation:ofType:` (page 73)

**Declared In**
`NSDocument.h`

## initWithContentsOfFile:ofType:

Initializes and returns an `NSDocument` object. (Deprecated in Mac OS X v10.4. Use `initWithContentsOfURL:ofType:error:` (page 32) instead.)

```
- (id)initWithContentsOfFile:(NSString *)fileName ofType:(NSString *)docType
```

**Discussion**
Initializes and returns an NSDocument object of document type *docType* containing data stored in the file *fileName*. In opening the file, invokes the `readFromFile:ofType:` (page 74) method. If the document successfully opens the file, it calls setFileName: and `setFileType:` (page 51) with *fileName* and *docType*, respectively, as arguments. If the file cannot be opened, or the document is unable to load the contents of the file, this method returns `nil`. This initializer is typically invoked by the `NSDocumentController` method `makeDocumentWithContentsOfFile:ofType:`.

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

Deprecated in Mac OS X v10.4.

**Related Sample Code**
QTAudioExtractionPanel

QTKitAdvancedDocument

QTKitImport

QTKitPlayer

**Declared In**
`NSDocument.h`

## initWithContentsOfURL:ofType:

Initializes and returns an NSDocument object of a given document type. (Deprecated in Mac OS X v10.4. Use `initWithContentsOfURL:ofType:error:` (page 32) instead.)

```
- (id)initWithContentsOfURL:(NSURL *)aURL ofType:(NSString *)docType
```

**Discussion**

Initializes and returns an `NSDocument` object of document type *`docType`* containing data stored at *`aURL`*. In opening the location, invokes the `readFromURL:ofType:` (page 74) method. If the document successfully opens the location, it calls setFileName: and `setFileType:` (page 51) with the location's path and *`docType`*, respectively, as arguments. If the location cannot be opened, or the document is unable to load the contents of the location, this method returns `nil`. This initializer is typically invoked by the `NSDocumentController` method `makeDocumentWithContentsOfURL:ofType:`.

**Availability**

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

**Declared In**

`NSDocument.h`

## loadDataRepresentation:ofType:

Overridden by subclasses to load document data. (Deprecated in Mac OS X v10.4. Use `readFromData:ofType:error:` (page 40) instead.)

```
- (BOOL)loadDataRepresentation:(NSData *)docData ofType:(NSString *)docType
```

**Discussion**

Overridden by subclasses to load document data (*`docData`*) of type *`docType`* into the receiver, display it in windows, and return whether the operation was successful. This method is typically invoked by `loadFileWrapperRepresentation:ofType:` (page 73) after an `NSData` object is created from the contents of the file wrapper (which can include directories). The default implementation `raises` an `NSInternalInconsistencyException`. Subclasses must override this method unless they override `readFromFile:ofType:` (page 74) or `loadFileWrapperRepresentation:ofType:` (page 73) to do specialized reading and loading of document data.

The *`docType`* argument is the type name corresponding to the value of the `CFBundleTypeName` entry in the document type's `Info.plist` dictionary.

Here is an example implementation:

```
//Document type name
NSString *MyDocumentType = @"Rich Text Format (RTF) document";

...

- (BOOL)loadDataRepresentation:(NSData *)data ofType:(NSString *)aType  {
    NSAssert([aType isEqualToString: MyDocumentType], @"Unknown  type");
    fileContents = [data copyWithZone:[self zone]];
    return YES;
}
```

**Availability**

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

**See Also**

`- dataRepresentationOfType:` (page 69)

**Declared In**
`NSDocument.h`

## loadFileWrapperRepresentation:ofType:

Loads document data from a given file wrapper. (Deprecated in Mac OS X v10.4. Use `readFromFileWrapper:ofType:error:` (page 41) instead.)

```
- (BOOL)loadFileWrapperRepresentation:(NSFileWrapper *)wrapper ofType:(NSString
    *)docType
```

**Discussion**
Loads document data in file wrapper *wrapper* of type *docType* into the receiver, displays it in windows, and returns whether the operation was successful. If *wrapper* is a simple file, it invokes `loadDataRepresentation:ofType:` (page 72) to load the data. If *wrapper* is a directory, it returns `NO` by default; subclasses can override to handle file wrappers that are directories. This method is typically invoked by `readFromFile:ofType:` (page 74) after it creates an NSData object from the contents of the file.

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

Deprecated in Mac OS X v10.4.

**See Also**
`- fileWrapperRepresentationOfType:` (page 70)

**Declared In**
`NSDocument.h`

## printShowingPrintPanel:

Overridden by subclasses to print the current document's (the receiver's) data. (Deprecated in Mac OS X v10.4. Use `printDocumentWithSettings:showPrintPanel:delegate:didPrintSelector:contextInfo:` (page 38) instead.)

```
- (void)printShowingPrintPanel:(BOOL)flag
```

**Discussion**
Overridden by subclasses to print the current document's (the receiver's) data; if *flag* is `YES`, the implementation should first display the Print panel. This method is typically invoked by `printDocument:` with an argument of `YES`. The default implementation does nothing. If there is any printing information other than that encoded in the receiver's `NSPrintInfo` object, subclasses should get it here.

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

Deprecated in Mac OS X v10.4.

**See Also**
`- printInfo` (page 39)

**Declared In**
`NSDocument.h`

## readFromFile:ofType:

Reads and loads document data of the given type from the given file. (Deprecated in Mac OS X v10.4. Use `readFromURL:ofType:error:` (page 42) instead.)

```
- (BOOL)readFromFile:(NSString *)fileName ofType:(NSString *)docType
```

**Discussion**
Reads and loads document data of type *docType* from the file *fileName*, returning whether the operation was successful. This method invokes `loadDataRepresentation:ofType:` and is invoked when the receiver is first created and initialized by `initWithContentsOfFile:ofType:`. It uses `NSData initWithContentsOfFile:` to get the document data.

This method is one of the location-based primitives. Subclasses can override this method instead of overriding `loadDataRepresentation:ofType:` to read and load document data. Subclasses that handle file packages such as RTFD or that treat locations of files as anything other than paths should override this method. Override implementations of this method can filter the document data using `NSPasteboard` or other filtering services.

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

Deprecated in Mac OS X v10.4.

**See Also**
`- dataRepresentationOfType:` (page 69)
`- writeToFile:ofType:` (page 77)

**Declared In**
`NSDocument.h`

## readFromURL:ofType:

Reads and loads document data. (Deprecated in Mac OS X v10.4. Use `readFromURL:ofType:error:` (page 42) instead.)

```
- (BOOL)readFromURL:(NSURL *)aURL ofType:(NSString *)docType
```

**Discussion**
Reads and loads document data of type *docType* from the URL *aURL*, returning whether the operation was successful. This method only supports URLs of the `file:` scheme and calls `readFromFile:ofType:` (page 74).

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

Deprecated in Mac OS X v10.4.

**Declared In**
`NSDocument.h`

## revertToSavedFromFile:ofType:

Reverts the receiver to the data stored in the file system. (Deprecated in Mac OS X v10.4. Use `revertToContentsOfURL:ofType:error:` (page 43) instead.)

```
- (BOOL)revertToSavedFromFile:(NSString *)fileName ofType:(NSString *)type
```

**Discussion**
Reverts the receiver to the data stored in the file system in file named *fileName* of file type *type*. Invokes `readFromFile:ofType:` (page 74) and returns whether that method successfully read the file and processed the document data.

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

Deprecated in Mac OS X v10.4.

**See Also**
`- revertDocumentToSaved:` (page 43)

**Declared In**
`NSDocument.h`

## revertToSavedFromURL:ofType:

Reverts the receiver. (Deprecated in Mac OS X v10.4. Use `revertToContentsOfURL:ofType:error:` (page 43) instead.)

```
- (BOOL)revertToSavedFromURL:(NSURL *)aURL ofType:(NSString *)type
```

**Discussion**
Reverts the receiver to the data stored at *aURL* of type *type*. Invokes `readFromURL:ofType:` (page 74) and returns whether that method successfully read the file and processed the document data.

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

Deprecated in Mac OS X v10.4.

**See Also**
`- revertDocumentToSaved:` (page 43)

**Declared In**
`NSDocument.h`

## runModalPageLayoutWithPrintInfo:

Runs the page layout modal panel with the receiver's printing information object. (Deprecated in Mac OS X v10.4. Use `runModalPageLayoutWithPrintInfo:delegate:didRunSelector:contextInfo:` (page 44) instead.)

```
- (NSInteger)runModalPageLayoutWithPrintInfo:(NSPrintInfo *)printInfo
```

**Discussion**
Runs the page layout modal panel with the receiver's printing information object (`printInfo`) as argument and returns the result constant (indicating the button pressed by the user). To run as sheet on the receiver's document window, use `runModalPageLayoutWithPrintInfo:delegate:didRunSelector:contextInfo:` (page 44) instead.

**Availability**
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

**See Also**
`- shouldChangePrintInfo:` (page 55)
`- runModalWithPrintInfo:` (NSPageLayout)

**Declared In**
`NSDocument.h`

## saveToFile:saveOperation:delegate:didSaveSelector:contextInfo:

Called after the user has been given the opportunity to select a destination through the modal Save panel. (Deprecated in Mac OS X v10.4. Use `saveToURL:ofType:forSaveOperation:delegate:didSaveSelector:contextInfo:` (page 49) instead.)

```
- (void)saveToFile:(NSString *)fileName
    saveOperation:(NSSaveOperationType)saveOperation delegate:(id)delegate
    didSaveSelector:(SEL)didSaveSelector contextInfo:(void *)contextInfo
```

**Discussion**
Called after the user has been given the opportunity to select a destination through the modal Save panel presented by `runModalSavePanelForSaveOperation:delegate:didSaveSelector:contextInfo:` (page 45). The *`delegate`* is assigned to the Save panel. If *`fileName`* is non-`nil`, this method writes the document to *`fileName`*, sets the document's file location and document type (if a native type), and clears the document's edited status. *`didSaveSelector`* gets called with `YES` if the document is saved successfully, and `NO` otherwise. The *`saveOperation`* is one of the constants in "Constants" (page 66). Pass *`contextInfo`* with the callback.

The *`didSaveSelector`* callback method should have the following signature:

```
- (void)document:(NSDocument *)doc didSave:(BOOL)didSave contextInfo:(void
*)contextInfo
```

**Availability**
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

**Declared In**
`NSDocument.h`

## setFileName:

Sets the file (filename and directory path) under which document data is saved. (Deprecated in Mac OS X v10.4. Use `setFileURL:` (page 52) instead.)

```
- (void)setFileName:(NSString *)fileName
```

**Discussion**
Sets the file (filename and directory path) under which document data is saved to *fileName*. As a side effect, synchronizes the titles of the document's windows with the new name or location. A document's filename is automatically set when it is saved as a new document (Save) and when an existing document is saved under a different filename or path (Save As). The Finder also keeps track of open documents and their associated files. When a user moves or renames a file in the Finder that corresponds to an open document, the Finder calls `setFileName:` with the new filename.

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

Deprecated in Mac OS X v10.4.

**See Also**
- `fileName` (page 70)

**Declared In**
`NSDocument.h`

## writeToFile:ofType:

Writes document data to a file. (Deprecated in Mac OS X v10.4. Use `writeToURL:ofType:error:` (page 64) instead.)

```
- (BOOL)writeToFile:(NSString *)fileName ofType:(NSString *)docType
```

**Discussion**
Writes document data of type *docType* to the file *fileName*, returning whether the operation was successful. This method invokes `dataRepresentationOfType:` (page 69) and is indirectly invoked whenever the document file is saved. It uses the `NSData` method `writeToFile:atomically:` to write to the file.

This method is one of the location-based primitives. Subclasses can override this method instead of overriding `dataRepresentationOfType:` to write document data to the file system as an `NSData` object after creating that object from internal data structures. Subclasses that handle file packages such as RTFD or that treat locations of files as anything other than paths should override this method. Override implementations of this method should ensure that they filter document data appropriately using `NSPasteboard` filtering services.

See "NSDocument Saving Behavior" (page 9) for additional information about saving documents.

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

Deprecated in Mac OS X v10.4.

**See Also**
- `loadDataRepresentation:ofType:` (page 72)
- `readFromFile:ofType:` (page 74)
- `writeToFile:ofType:originalFile:saveOperation:` (page 78)

**Declared In**
`NSDocument.h`

## writeToFile:ofType:originalFile:saveOperation:

Writes the receiver document's contents to a file. (Deprecated in Mac OS X v10.4. Use `writeToURL:ofType:forSaveOperation:originalContentsURL:error:` (page 65) instead.)

```
- (BOOL)writeToFile:(NSString *)fullDocumentPath ofType:(NSString *)docType
    originalFile:(NSString *)fullOriginalDocumentPath
    saveOperation:(NSSaveOperationType)saveOperationType
```

**Discussion**
This method is called from `writeWithBackupToFile:ofType:saveOperation:` (page 78) to actually write the file of type *docType* to *fullDocumentPath*. *fullOriginalDocumentPath* is the path to the original file if there is one and `nil` otherwise. The default implementation simply calls `writeToFile:ofType:` (page 77). You should not need to call this method directly, but subclasses that need access to the previously saved copy of their document while saving the new one can override this method. The *saveOperationType* argument is one of the constants listed in "Constants" (page 66).

See "NSDocument Saving Behavior" (page 9) for additional information about saving documents.

**Availability**
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

**Declared In**
`NSDocument.h`

## writeToURL:ofType:

Writes document data to a URL. (Deprecated in Mac OS X v10.4. Use `writeToURL:ofType:error:` (page 64) instead.)

```
- (BOOL)writeToURL:(NSURL *)aURL ofType:(NSString *)docType
```

**Discussion**
Writes document data of type *docType* to the URL *aURL*, returning whether the operation was successful. This method only supports URLs of the `file:` scheme and calls `writeToFile:ofType:` (page 77).

**Availability**
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

**Declared In**
`NSDocument.h`

## writeWithBackupToFile:ofType:saveOperation:

This method is called by action methods to save document contents to a file. (Deprecated in Mac OS X v10.4. Use `writeSafelyToURL:ofType:forSaveOperation:error:` (page 63) instead.)

```
- (BOOL)writeWithBackupToFile:(NSString *)fullDocumentPath ofType:(NSString *)docType
    saveOperation:(NSSaveOperationType)saveOperationType
```

**Discussion**
This method is called by action methods like `saveDocument:` (page 47), `saveDocumentAs:` (page 47), and `saveDocumentTo:` (page 48). It is responsible for handling backup of the existing file, if any, and removal of that backup if `keepBackupFile` (page 34) returns `NO`. In between those two things, it calls `writeToFile:ofType:originalFile:saveOperation:` (page 78) to write the document of type *`docType`* to *`fullDocumentPath`*. You should never need to call `writeWithBackupToFile:ofType:saveOperation:`, but subclasses that want to change the way the backup works can override it. The *`saveOperationType`* argument is one of the constants listed in "Constants" (page 66).

If you override this method, you should invoke `fileAttributesToWriteToFile:ofType:saveOperation:` (page 70) and set the variables returned from this method when writing *`fullDocumentPath`*. `NSFileManager` `changeFileAttributes:atPath:` can be used to do this.

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

Deprecated in Mac OS X v10.4.

**Related Sample Code**
QTMetadataEditor

**Declared In**
`NSDocument.h`

# Document Revision History

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

| Date | Notes |
|---|---|
| 2009-01-06 | Added description of NSChangeRedone constant. Updated other constant declarations and revised their descriptions. Added subclassing advice section with cross reference to main discussion. |
| 2008-03-11 | Revised descriptions of saveDocumentWithDelegate:didSaveSelector: contextInfo: and isDocumentEdited methods. Added descriptions (previously removed) of deprecated methods no longer appearing in header file. |
| 2007-04-03 | Added description of method introduced in Mac OS X v10.5. Revised task categories. Removed comment stating incorrectly that close method is not called when quitting an application. Removed descriptions of deprecated methods no longer defined in header file. |
| 2006-07-24 | Rewrote description of runModalSavePanelForSaveOperation:delegate: didSaveSelector:contextInfo: method; reformatted parameter descriptions and made minor revisions throughout. |
| 2006-05-23 | First publication of this content as a separate document. |

# Index