

[AppKit](#) / [NSDocument](#)

Class

NSDocument

An abstract class that defines the interface for macOS documents.

macOS

```
@MainActor
class NSDocument
```

Overview

A document is an object that can internally represent data displayed in a window and that can read data from and write data to a file or file package. 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.

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 AppKit, 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 AppKit classes that establish an architectural basis for document-based apps (the others being [NSDocumentController](#) and [NSWindowController](#)).

For more information about using [NSDocument](#) in a document-based app, see [Developing a Document-Based App](#).

Subclassing NSDocument

The [NSDocument](#) class is designed to be subclassed. That is, the [NSDocument](#) class is abstract, and your app must create at least one [NSDocument](#) subclass in order 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 [Document-Based App Programming Guide for Mac](#) 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.

Document Saving Behavior

The [NSDocument](#) class 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 macOS. It is important that overrides of [write\(to:ofType:\)](#) and [write\(to:ofType:for:originalContentsURL:\)](#) 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 value in [fileURL](#).

When updating your app to link against OS X v10.5, keep in mind that it is usually more appropriate to invoke in your app code one of the `NSDocument` `save...` methods than one of the `write...` methods. The `write...` methods are there primarily for you to override. The `saveToURL:ofType:forSaveOperation:error:` method that is meant always to be invoked during document saving, sets the `fileModificationDate` property with the file's new modification date after it has been written (for `NSDocument.SaveOperationType.saveOperation` and `NSDocument.SaveOperationType.saveAsOperation` only).

Likewise, it's usually more appropriate to invoke in your app code one of the `NSDocument` `revert...` methods than one of the `read...` methods. The `read...` methods are there primarily for you to override. The `revert(toContentsOf:ofType:)` method that is meant always to be invoked during rereading of an open document, sets the `fileModificationDate` property with the file's modification date after it has been read.

iCloud Support

The `NSDocument` class implements the file coordination support that is required for an iCloud-enabled, document-based Mac app (see [How iCloud Document Storage Works](#) in [iCloud Design Guide](#)). In addition, this class's methods for moving and renaming documents, new in OS X v10.8, ensure that these operations are performed in a safe manner for iCloud-enabled apps.

Multicore Considerations

In macOS 10.6 and later, `NSDocument` supports the ability to open multiple documents concurrently. However, this support requires the cooperation of the document object. If your document subclass is able to read specific document types independently of other similar documents, you should override the `canConcurrentlyReadDocuments(ofType:)` class method and return `true` for the appropriate document types. If specific document types rely on shared state information, however, you should return `false` for those types.

Topics

Creating a Document Object

`init()`

Initializes and returns an empty document object.

`convenience init(contentsOf: URL, ofType: String)` throws

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

`convenience init(for: URL?, withContentsOf: URL, ofType: String) throws`
Initializes a document with the specified contents, and places the resulting document's file at the designated location.

`convenience init(type: String) throws`
Initializes a document of a specified type.

Reading the Document's Content

`class func canConcurrentlyReadDocuments(ofType: String) -> Bool`
Returns a Boolean value that indicates whether the receiver reads multiple documents of the given type concurrently.

`func read(from: URL, ofType: String) throws`
Sets the contents of this document by reading from a file or file package, of a specified type, located by a URL.

`func read(from: FileWrapper, ofType: String) throws`
Sets the contents of this document by reading from a file wrapper of a specified type.

`func read(from: Data, ofType: String) throws`
Sets the contents of this document by reading from data of a specified type.

Writing the Document's Content

`func canAsynchronouslyWrite(to: URL, ofType: String, for: NSDocument.SaveOperationType) -> Bool`
Returns whether the receiver can concurrently write to a file or file package located by a URL, that is formatted for a specific type, for a specific kind of save operation.

`func unblockUserInteraction()`
Unblocks the main thread during asynchronous saving.

`func write(to: URL, ofType: String) throws`
Writes the contents of the document to a file or file package located by a URL, that is formatted to a specified type.

`func writeSafely(to: URL, ofType: String, for: NSDocument.SaveOperationType) throws`
Writes the contents of the document to a file or file package located by a URL.

```
func fileWrapper(ofType: String) throws -> FileWrapper
```

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

```
func data(ofType: String) throws -> Data
```

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

```
func write(to: URL, ofType: String, for: NSDocument.SaveOperationType, originalContentsURL: URL?) throws
```

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

```
func save(to: URL, ofType: String, for: NSDocument.SaveOperationType, delegate: Any?, didSave: Selector?, contextInfo: UnsafeMutableRaw Pointer?)
```

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

```
func save(to: URL, ofType: String, for: NSDocument.SaveOperationType, completionHandler: ((any Error?) -> Void)
```

Saves the contents of the document to a file or file package located by a URL, that is formatted to a specified type, for a particular kind of save operation, and invokes the passed-in completion handler.

```
func fileAttributesToWrite(to: URL, ofType: String, for: NSDocument.SaveOperationType, originalContentsURL: URL?) throws -> [String : Any]
```

Returns the attributes to write to the file or file package at the specified URL, and targeting the specified type of save operation.

```
enum SaveOperationType
```

Constants for specifying the type of document-save operation to perform.

Getting Document Metadata

```
var fileURL: URL?
```

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

```
var isEntireFileLoaded: Bool
```

A Boolean value that indicates whether the document's file is completely loaded into memory.

```
var fileModificationDate: Date?
```

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

```
var keepBackupFile: Bool
```

A Boolean value that indicates whether the document archives previously saved versions of the document.

```
var isDraft: Bool
```

A Boolean value that indicates whether the document is a draft that the user has not yet saved.

```
var fileType: String?
```

The name of the document type, as specified in the app's information property-list file.

```
var isDocumentEdited: Bool
```

A Boolean value that indicates whether the document has unsaved changes.

```
var isInViewingMode: Bool
```

A Boolean value that indicates whether the document is in read-only mode.

Managing File Type Information

```
class var readableTypes: [String]
```

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

```
class var writableTypes: [String]
```

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

```
class func isNativeType(String) -> Bool
```

Returns a Boolean value that indicates whether the document can read and write the data natively.

```
func writableTypes(for: NSDocument.SaveOperationType) -> [String]
```

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

```
func fileNameExtension(forType: String, saveOperation: NSDocument.SaveOperationType) -> String?
```

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

Creating and Managing Window Controllers

```
func makeWindowControllers()
```

Creates the window controller objects that the document uses to display its content.

```
func addWindowController(NSWindowController)
```

Adds the specified window controller to the current document.

```
func removeWindowController(NSWindowController)
```

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

```
var windowControllers: [NSWindowController]
```

The document's current window controllers.

```
var windowNibName: NSNib.Name?
```

The name of the document's sole nib file.

```
func windowControllerDidLoadNib(NSWindowController)
```

Called after one of the document's window controllers loads its nib file.

```
func windowControllerWillLoadNib(NSWindowController)
```

Called before one of the document's window controllers loads its nib file.

```
func shouldCloseWindowController(NSWindowController, delegate: Any?,  
shouldClose: Selector?, contextInfo: UnsafeMutableRawPointer?)
```

Determines whether the system should close the document and its associated window.

Managing Document Windows

```
func showWindows()
```

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

```
func setWindow(NSWindow?)
```

Sets the window outlet of this document to the specified value.

```
var windowForSheet: NSWindow?
```

Returns the document window to use as the parent of a document-modal sheet.

```
var displayName: String!
```

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

```
func defaultDraftName() -> String
```

Returns the default draft name for the document subclass.

```
func encodeRestorableState(with: NSCoder, backgroundQueue: Operation Queue)
```

Saves the interface-related state of the document.

Configuring the Autosave Behavior

```
class var autosavesInPlace: Bool
```

A Boolean value that indicates whether the document subclass supports autosaving in place.

```
class var autosavesDrafts: Bool
```

A Boolean value that indicates whether the document subclass supports autosaving of drafts.

```
class var preservesVersions: Bool
```

A Boolean value that indicates whether the document subclass supports version management.

```
var autosavedContentsFileURL: URL?
```

The location of the most recently autosaved document contents.

```
var autosavingFileType: String?
```

The document type to use for an autosave operation.

```
var autosavingIsImplicitlyCancellable: Bool
```

A Boolean value that indicates whether you can cancel an in-progress autosave operation.

Autosaving the Document

```
func checkAutosavingSafety() throws
```

Returns a Boolean value that indicates whether it is safe to autosave document changes.

```
var hasUnautosavedChanges: Bool
```

A Boolean value that indicates whether the document has changes that have not been autosaved.

```
func scheduleAutosaving()
```

Schedules periodic autosaving for the purpose of crash protection.

```
func autosave(withDelegate: Any?, didAutosave: Selector?, contextInfo: UnsafeMutableRawPointer?)
```

Autosaves the document's contents to an appropriate location in the file system.

```
func autosave(withImplicitCancellability: Bool, completionHandler:
((any Error)?) -> Void)
```

Autosaves the document's contents to an appropriate file-system location, as needed.

```
var backupFileURL: URL?
```

The URL for the document's backup file that was created during an autosave operation.

Browsing Document Versions

```
func browseVersions(Any?)
```

Opens the Versions browser in the document's main window.

```
var isBrowsingVersions: Bool
```

A Boolean value that indicates whether the document is currently displaying the Versions browser.

```
func stopBrowsingVersions(completionHandler: (() -> Void)?)
```

Dismiss the Versions browser for the current document.

Storing Documents in iCloud

```
func moveToUbiquityContainer(Any?)
```

Moves the document to the user's iCloud storage.

```
class var usesUbiquitousStorage: Bool
```

Returns whether the document object stores its contents in the user's iCloud document storage.

Managing Undo and Redo Actions

```
var undoManager: UndoManager?
```

The object that the document uses to support undo/redo operations.

```
var hasUndoManager: Bool
```

A Boolean value that indicates whether the document owns an undo manager object.

Updating the Document Change Count

```
func updateChangeCount(withToken: Any, for: NSDocument.SaveOperationType)
```

Updates the document's change count settings after a successful save operation.

```
func updateChangeCount(NSDocument.ChangeType)
```

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

```
enum ChangeType
```

Values that indicate a document's edit status.

```
func changeCountToken(for: NSDocument.SaveOperationType) -> Any
```

Returns an object that encapsulates the current record of document changes at the beginning of a save operation.

Handling Window Restoration

```
class func allowedClasses(forRestorableStateKeyPath: String) -> [AnyClass]
```

Returns the classes that support secure coding.

```
func encodeRestorableState(with: NSCoder)
```

Saves the interface-related state of the document.

```
func restoreState(with: NSCoder)
```

Restores the interface-related state of the document.

```
class var restorableStateKeyPaths: [String]
```

Returns an array of key paths that represent the restorable attributes of the document.

```
func invalidateRestorableState()
```

Marks the document's interface-related state as dirty.

```
func restoreWindow(withIdentifier: NSUserInterfaceItemIdentifier, state: NSCoder, completionHandler: (NSWindow?, (any Error)?) -> Void)
```

Restores a window that was associated with a document, after that document is reopened.

Presenting a Save Panel

```
func runModalSavePanel(for: NSDocument.SaveOperationType, delegate: Any?, didSave: Selector?, contextInfo: UnsafeMutableRawPointer?)
```

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

```
func prepareSavePanel(NSSavePanel) -> Bool
```

Tells the document to customize the specified Save panel.

```
var shouldRunSavePanelWithAccessoryView: Bool
```

A Boolean value that indicates whether the document's Save panel displays a list of supported writable document types.

Deprecated

```
var fileTypeFromLastRunSavePanel: String?
```

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

```
var fileNameExtensionWasHiddenInLastRunSavePanel: Bool
```

A Boolean value that indicates whether the user chose to hide the document's filename extension.

Supporting User Activities

```
var userActivity: NSUserActivity?
```

An object that encapsulates a user activity the document supports.

```
func updateUserActivityState(NSUserActivity)
```

Updates the state of the given user activity.

```
let NSUserActivityDocumentURLKey: String
```

The key that identifies the document associated with a user activity.

Validating User Interface Items

```
func validateUserInterfaceItem(any NSValidatedUserInterfaceItem) -> Bool
```

Validates the specified user interface item that the receiver manages.

Performing Tasks Serially

```
func performSynchronousFileAccess(() -> Void)
```

Waits for any scheduled file access to complete, then invokes the passed-in block.

```
func performAsynchronousFileAccess(() -> Void) -> Void)
```

Waits for any scheduled file access to complete but without blocking the main thread, then invokes the passed-in block.

```
func performActivity(withSynchronousWaiting: Bool, using: (() -> Void) -> Void)
```

Waits for any work scheduled by previous invocations of this method to complete, then invokes the passed-in block.

```
func continueActivity(() -> Void)
```

Continues to perform the task for a user activity object using a different block.

```
func continueAsynchronousWorkOnMainThread(() -> Void)
```

Invokes the passed-in block on the main thread.

Handling User Actions

```
func printDocument(Any?)
```

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

```
func runPageLayout(Any?)
```

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

```
func revertToSaved(Any?)
```

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

```
func save(Any?)
```

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

```
func saveAs(Any?)
```

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

```
func saveTo(Any?)
```

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

```
func save(withDelegate: Any?, didSave: Selector?, contextInfo: UnsafeMutableRawPointer?)
```

Saves the document and delivers the results to the provided delegate object.

Closing the Document

```
func canClose(withDelegate: Any, shouldClose: Selector?, contextInfo: UnsafeMutableRawPointer?)
```

Determines whether to close the document, prompting the user as needed to choose a course of action.

```
func close()
```

Closes all of the document's windows and removes the document from its document controller.

Reverting the Document Contents

```
func revert(toContentsOf: URL, ofType: String) throws
```

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.

Duplicating the Document

```
func duplicate() throws -> NSDocument
```

Creates a new document whose contents are the same as the receiver and returns an error object if unsuccessful.

```
func duplicate(Any?)
```

Creates a copy of the receiving document in response to the user choosing Duplicate from the File menu.

```
func duplicate(withDelegate: Any?, didDuplicate: Selector?, contextInfo: UnsafeMutableRawPointer?)
```

Creates a new document whose contents are the same as the current document.

Renaming the Document

```
func rename(Any?)
```

Renames the current document in response to the user choosing the Rename menu item.

Moving the Document

```
func move(Any?)
```

Moves the document to a new location in response to the user choosing the Move To... menu item.

```
func move(completionHandler: ((Bool) -> Void)?)
```

Moves the document to a user-selected location.

```
func move(to: URL, completionHandler: (((any Error)?) -> Void)?)
```

Moves the document's file to the given URL.

Locking the Document

```
func lock(Any?)
```

Locks the document in response to the user choosing the Lock menu item.

```
func unlock(Any?)
```

Unlocks the document in response to the user choosing the Unlock menu item.

```
func lock(completionHandler: ((Bool) -> Void)?)
```

Prevents the user from making further changes to the document.

```
func lock(completionHandler: (((any Error)?) -> Void)?)
```

Prevents the user from making changes to the document's file.

```
func unlock(completionHandler: ((Bool) -> Void)?)
```

Allows the user to make modifications to the document.

```
func unlock(completionHandler: (((any Error)?) -> Void)?)
```

Allows the user to make modifications to the document's file.

```
var isLocked: Bool
```

A Boolean value that indicates whether or not the file can be written to.

Printing the Document

```
var printInfo: NSPrintInfo
```

The printing information associated with the document.

```
func preparePageLayout(NSPageLayout) -> Bool
```

Adds document-specific content to the Page Layout panel.

```
func runModalPageLayout(with: NSPrintInfo, delegate: Any?, didRun: Selector?, contextInfo: UnsafeMutableRawPointer?)
```

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

```
func runModalPrintOperation(NSPrintOperation, delegate: Any?, didRun: Selector?, contextInfo: UnsafeMutableRawPointer?)
```

Runs the specified print operation modally.

```
func shouldChangePrintInfo(NSPrintInfo) -> Bool
```

Returns a Boolean value that indicates whether the document allows changes to the default printing information.

```
func print(withSettings: [NSPrintInfo.AttributeKey : Any], showPrint Panel: Bool, delegate: Any?, didPrint: Selector?, contextInfo: UnsafeMutableRawPointer?)
```

Prints the document's contents, optionally displaying a print panel to the user.

```
func printOperation(withSettings: [NSPrintInfo.AttributeKey : Any]) throws -> NSPrintOperation
```

Creates and returns a print operation for the document's contents.

```
var pdfPrintOperation: NSPrintOperation
```

A print operation you can use to create a PDF representation of the document's current contents.

```
func saveToPDF(Any?)
```

Exports a PDF representation of the document's current contents.

Sharing the Document

```
var allowsDocumentSharing: Bool
```

A Boolean value that indicates whether the document is shareable from the standard Share menu.

```
func prepare(NSSharingServicePicker)
```

Perform any custom setup associated with a sharing service picker.

```
func share(with: NSSharingService, completionHandler: ((Bool) -> Void)?)
```

Share the document's file using the specified sharing service.

Handling Script Commands

```
func handleClose(NSCloseCommand) -> Any?
```

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

```
func handlePrint(NSScriptCommand) -> Any?
```

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

```
func handleSave(NSScriptCommand) -> Any?
```

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

```
var objectSpecifier: NSScriptObjectSpecifier
```

Returns the object specifier that represents the document.

```
var lastComponentOfFileName: String
```

The name of the document seen by the user in AppleScript.

Displaying Errors to the User

```
func presentError(any Error, modalFor: NSWindow, delegate: Any?, didPresent: Selector?, contextInfo: UnsafeMutableRawPointer?)
```

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

```
func presentError(any Error) -> Bool
```

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

```
func willPresentError(any Error) -> any Error
```

Called when the receiver is about to present an error.

```
func willNotPresentError(any Error)
```

Confirms that the error object is not to be presented to the user and the error cannot be recovered from, so cleanup can be done.

Deprecated

Avoid using deprecated classes and protocols in your apps.

⋮ Deprecated Symbols

Review symbols that are no longer supported, and find replacements.

Instance Properties

```
var observedPresentedItemUbiquityAttributes: Set<URLResourceKey>

var presentedItemURL: URL?

var previewRepresentableActivityItems: [any NSPreviewRepresentable
ActivityItem]?

var savePanelShowsFileFormatsControl: Bool
```

Instance Methods

```
func accommodatePresentedItemDeletion(completionHandler: ((any Error)?)
-> Void)

func presentedItemDidChange()

func presentedItemDidChangeUbiquityAttributes(Set<URLResourceKey>)

func presentedItemDidGain(NSFileVersion)

func presentedItemDidLose(NSFileVersion)

func presentedItemDidMove(to: URL)

func presentedItemDidResolveConflict(NSFileVersion)

func relinquishPresentedItem(toReader: (() -> Void)? ) -> Void)

func relinquishPresentedItem(toWriter: (() -> Void)? ) -> Void)

func savePresentedItemChanges(completionHandler: ((any Error)? ) -> Void
)
```

Relationships

Inherits From

NSObject

Inherited By

NSPersistentDocument

Conforms To

CVarArg
Copyable
CustomDebugStringConvertible
CustomStringConvertible
Equatable
Hashable
NSEditorRegistration
NSFilePresenter
NSMenuItemValidation
NSObjectProtocol
NSUserActivityRestoring
NSUserInterfaceValidations
Sendable

See Also

Documents

`{}` Developing a Document-Based App
Write an app that creates, manages, edits, and saves text documents.

`class` `NSDocumentController`
An object that manages an app's documents.

`class` `NSPersistentDocument`
A document object that can integrate with Core Data.