NSApplication Class Reference

Cocoa > Runtime Architecture

2009-02-04

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

NSApplication Class Reference

Inherits from	NSResponder : NSObject
Conforms to	NSUserInterfaceValidations NSCoding (NSResponder) NSObject (NSObject)
Framework	/System/Library/Frameworks/AppKit.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	NSApplication.h NSApplicationScripting.h NSColorPanel.h NSHelpManager.h NSPageLayout.h
Companion guides	Application Architecture Overview Notification Programming Topics for Cocoa Sheet Programming Topics for Cocoa System Services
Related sample code	CoreRecipes ImageClient NumberInput_IMKit_Sample Quartz Composer WWDC 2005 TextEdit TextEditPlus

Class at a Glance

An `NSApplication` object manages an application's main event loop in addition to resources used by all of that application's objects.

Principal Attributes

- Delegate
- Key window
- Display context
- List of windows
- Main window

Commonly Used Methods

`keyWindow` (page 35)
: Returns an `NSWindow` object representing the key window.

`mainWindow` (page 36)
: Returns the application's main window.

`registerServicesMenuSendTypes:returnTypes:` (page 42)
: Specifies which services are valid for this application.

`runModalForWindow:` (page 46)
: Runs a modal event loop for the specified `NSWindow` object.

Overview

The `NSApplication` class provides the central framework for your application's execution.

Every application must have exactly one instance of `NSApplication` (or a subclass of `NSApplication`). Your program's `main()` function should create this instance by invoking the `sharedApplication` (page 21) class method. After creating the `NSApplication` object, the `main()` function should load your application's main nib file and then start the event loop by sending the `NSApplication` object a `run` (page 45) message. If you create an Application project in Xcode, this `main()` function is created for you. The `main()` function Xcode creates begins by calling a function named `NSApplicationMain()`, which is functionally similar to the following:

```
void NSApplicationMain(int argc, char *argv[]) {
    [NSApplication sharedApplication];
    [NSBundle loadNibNamed:@"myMain" owner:NSApp];
    [NSApp run];
}
```

The `sharedApplication` (page 21) class method initializes the display environment and connects your program to the window server and the display server. The `NSApplication` object maintains a list of all the `NSWindow` objects the application uses, so it can retrieve any of the application's `NSView` objects. `sharedApplication` (page 21) also initializes the global variable `NSApp`, which you use to retrieve the `NSApplication` instance. `sharedApplication` (page 21) only performs the initialization once; if you invoke it more than once, it simply returns the `NSApplication` object it created previously.

`NSApplication` performs the important task of receiving events from the window server and distributing them to the proper `NSResponder` objects. `NSApp` translates an event into an `NSEvent` object, then forwards the `NSEvent` object to the affected `NSWindow` object. All keyboard and mouse events go directly to the `NSWindow` object associated with the event. The only exception to this rule is if the Command key is pressed when a key-down event occurs; in this case, every `NSWindow` object has an opportunity to respond to the event. When an `NSWindow` object receives an `NSEvent` object from `NSApp`, it distributes it to the objects in its view hierarchy.

`NSApplication` is also responsible for dispatching certain Apple events received by the application. For example, Mac OS X sends Apple events to your application at various times, such as when the application is launched or reopened. `NSApplication` installs Apple event handlers to handle these events by sending a message to the appropriate object. You can also use the `NSAppleEventManager` class to register your own Apple event handlers. The `applicationWillFinishLaunching:` (page 75) method is generally the best

place to do so. For more information on how events are handled and how you can modify the default behavior, including information on working with Apple events in scriptable applications, see How Cocoa Applications Handle Apple Events in *Cocoa Scripting Guide*.

The `NSApplication` class sets up autorelease pools (instances of the `NSAutoreleasePool` class) during initialization and inside the event loop—specifically, within its initialization (or `sharedApplication` (page 21)) and `run` (page 45) methods. Similarly, the methods the Application Kit adds to `NSBundle` employ autorelease pools during the loading of nib files. These autorelease pools aren't accessible outside the scope of the respective `NSApplication` and `NSBundle` methods. Typically, an application creates objects either while the event loop is running or by loading objects from nib files, so this lack of access usually isn't a problem. However, if you do need to use Cocoa classes within the `main()` function itself (other than to load nib files or to instantiate `NSApplication`), you should create an autorelease pool before using the classes and then release the pool when you're done. For more information, see `NSAutoreleasePool` in the *Foundation Framework Reference*.

The Delegate and Notifications

You can assign a delegate to `NSApp`. The delegate responds to certain messages on behalf of `NSApp`. Some of these messages, such as `application:openFile:` (page 63), ask the delegate to perform an action. Another message, `applicationShouldTerminate:` (page 73), lets the delegate determine whether the application should be allowed to quit. The `NSApplication` class sends these messages directly to its delegate.

The `NSApp` also posts notifications to the application's default notification center. Any object may register to receive one or more of the notifications posted by `NSApp` by sending the message `addObserver:selector:name:object:` to the default notification center (an instance of the `NSNotificationCenter` class). The delegate of `NSApp` is automatically registered to receive these notifications if it implements certain delegate methods. For example, `NSApp` posts notifications when it is about to be done launching the application and when it is done launching the application (`NSApplicationWillFinishLaunchingNotification` (page 86) and `NSApplicationDidFinishLaunchingNotification` (page 85)). The delegate has an opportunity to respond to these notifications by implementing the methods `applicationWillFinishLaunching:` (page 75) and `applicationDidFinishLaunching:` (page 69). If the delegate wants to be informed of both events, it implements both methods. If it needs to know only when the application is finished launching, it implements only `applicationDidFinishLaunching:` (page 69).

System Services

`NSApplication` interacts with the system services architecture to provide services to your application through the Services menu.

Subclassing Notes

You rarely should find a real need to create a custom `NSApplication` subclass. Unlike some object-oriented libraries, Cocoa does not require you to create a custom application class to customize application behavior. Instead it gives you many other ways to customize an application. This section discusses both some of the possible reasons to subclass `NSApplication` and some of the reasons *not* to subclass `NSApplication`.

To use a custom subclass of `NSApplication`, simply send `sharedApplication` (page 21) to your subclass rather than directly to `NSApplication`. If you create your application in Xcode, you can accomplish this by setting your custom application class to be the principal class. In Xcode, double-click the application target in the Groups and Files list to open the Info window for the target. Then display the Properties pane of the window and replace "NSApplication" in the Principal Class field with the name of your custom class. The `NSApplicationMain` function sends `sharedApplication` (page 21) to the principal class to obtain the global application instance (`NSApp`)—which in this case will be an instance of your custom subclass of `NSApplication`.

Important: Many Application Kit classes rely on the `NSApplication` class and may not work properly until this class is fully initialized. As a result, you should not, for example, attempt to invoke methods of other Application Kit classes from an initialization method of an `NSApplication` subclass.

Methods to Override

Generally, you subclass `NSApplication` to provide your own special responses to messages that are routinely sent to the global application object (`NSApp`). `NSApplication` does not have primitive methods in the sense of methods that you must override in your subclass. Here are four methods that are possible candidates for overriding:

- Override `run` (page 45) if you want the application to manage the main event loop differently than it does by default. (This a critical and complex task, however, that you should only attempt with good reason.)
- Override `sendEvent:` (page 49) if you want to change how events are dispatched or perform some special event processing.
- Override `requestUserAttention:` (page 45) if you want to modify how your application attracts the attention of the user (for example, offering an alternative to the bouncing application icon in the Dock).
- Override `targetForAction:` (page 55) to substitute another object for the target of an action message.

Special Considerations

The global application object uses autorelease pools in its `run` (page 45) method; if you override this method, you'll need to create your own autorelease pools.

Do not override `sharedApplication` (page 21). The default implementation, which is essential to application behavior, is too complex to duplicate on your own.

Alternatives to Subclassing

`NSApplication` defines over twenty delegate methods that offer opportunities for modifying specific aspects of application behavior. Instead of making a custom subclass of `NSApplication`, your application delegate may be able to implement one or more of these methods to accomplish your design goals. In general, a better design than subclassing `NSApplication` is to put the code that expresses your application's special behavior into one or more custom objects called controllers. Methods defined in your controllers can be invoked from a small dispatcher object without being closely tied to the global application object. For more about application architectures, see Cocoa Design Patterns and The Core Application Architecture.

Tasks

Getting the Application

`+ sharedApplication` (page 21)
Returns the application instance, creating it if it doesn't exist yet.

Configuring Applications

`- applicationIconImage` (page 24)
Returns the image used for the receiver's icon.

`- setApplicationIconImage:` (page 50)
Sets the receiver's icon to the specified image.

`- delegate` (page 29)
Returns the receiver's delegate.

`- setDelegate:` (page 51)
Makes the given object the receiver's delegate.

Launching Applications

`- finishLaunching` (page 32)
Activates the receiver, opens any files specified by the `NSOpen` user default, and unhighlights the application's icon.

`- applicationWillFinishLaunching:` (page 75) *delegate method*
Sent by the default notification center immediately before the application object is initialized.

`- applicationDidFinishLaunching:` (page 69) *delegate method*
Sent by the default notification center after the application has been launched and initialized but before it has received its first event.

Terminating Applications

`- terminate:` (page 57)
Terminates the receiver.

`- applicationShouldTerminate:` (page 73) *delegate method*
Sent to notify the delegate that the application is about to terminate.

`- applicationShouldTerminateAfterLastWindowClosed:` (page 74) *delegate method*
Invoked when the user closes the last window the application has open.

`- replyToApplicationShouldTerminate:` (page 43)
Responds to `NSTerminateLater` once the application knows whether it can terminate.

`- applicationWillTerminate:` (page 76) *delegate method*
Sent by the default notification center immediately before the application terminates.

Managing Active Status

- `isActive` (page 34)
  Returns a Boolean value indicating whether this is the active application.
- `activateIgnoringOtherApps:` (page 23)
  Makes the receiver the active application.
- `applicationWillBecomeActive:` (page 74) *delegate method*
  Sent by the default notification center immediately before the application becomes active.
- `applicationDidBecomeActive:` (page 68) *delegate method*
  Sent by the default notification center immediately after the application becomes active.
- `deactivate` (page 29)
  Deactivates the receiver.
- `applicationWillResignActive:` (page 76) *delegate method*
  Sent by the default notification center immediately before the application is deactivated.
- `applicationDidResignActive:` (page 70) *delegate method*
  Sent by the default notification center immediately after the application is deactivated.

Hiding Applications

- `hideOtherApplications:` (page 33)
  Hides all applications, except the receiver.
- `unhideAllApplications:` (page 59)
  Unhides all applications, including the receiver.
- `applicationWillHide:` (page 75) *delegate method*
  Sent by the default notification center immediately before the application is hidden.
- `applicationDidHide:` (page 69) *delegate method*
  Sent by the default notification center immediately after the application is hidden.
- `applicationWillUnhide:` (page 76) *delegate method*
  Sent by the default notification center immediately after the application is unhidden.
- `applicationDidUnhide:` (page 70) *delegate method*
  Sent by the default notification center immediately after the application is made visible.

Managing the Event Loop

- `isRunning` (page 34)
  Returns a Boolean value indicating whether the main event loop is running.
- `run` (page 45)
  Starts the main event loop.
- `stop:` (page 54)
  Stops the main event loop.
- `runModalForWindow:` (page 46)
  Starts a modal event loop for a given window.

- `stopModal` (page 55)
  Stops a modal event loop.
- `stopModalWithCode:` (page 55)
  Stops a modal event loop, allowing you to return a custom result code.
- `abortModal` (page 22)
  Aborts the event loop started by `runModalForWindow:` (page 46) or `runModalSession:` (page 47).
- `beginModalSessionForWindow:` (page 25)
  Sets up a modal session with the given window and returns an `NSModalSession` structure representing the session.
- `runModalSession:` (page 47)
  Runs a given modal session, as defined in a previous invocation of `beginModalSessionForWindow:`.
- `modalWindow` (page 37)
  Returns the modal window that the receiver is displaying.
- `endModalSession:` (page 31)
  Finishes a modal session.
- `sendEvent:` (page 49)
  Dispatches an event to other objects.

Handling Events

- `currentEvent` (page 28)
  Returns the current event, the last event the receiver retrieved from the event queue.
- `nextEventMatchingMask:untilDate:inMode:dequeue:` (page 37)
  Returns the next event matching a given mask, or `nil` if no such event is found before a specified expiration date.
- `discardEventsMatchingMask:beforeEvent:` (page 30)
  Removes all events matching the given mask and generated before the specified event.

Posting Events

- `postEvent:atStart:` (page 41)
  Adds a given event to the receiver's event queue.

Managing Sheets

- `beginSheet:modalForWindow:modalDelegate:didEndSelector:contextInfo:` (page 26)
  Starts a document modal session.
- `endSheet:` (page 31)
  Ends a document modal session by specifying the sheet window.
- `endSheet:returnCode:` (page 32)
  Ends a document modal session by specifying the sheet window.

Managing Windows

- `keyWindow` (page 35)
  Returns the window that currently receives keyboard events.
- `mainWindow` (page 36)
  Returns the main window.
- `windowWithWindowNumber:` (page 62)
  Returns the window corresponding to the specified window number.
- `windows` (page 61)
  Returns an array containing the receiver's window objects.
- `makeWindowsPerform:inOrder:` (page 36)
  Sends the specified message to each of the application's window objects until one returns a non-`nil` value.
- `applicationWillUpdate:` (page 77) *delegate method*
  Sent by the default notification center immediately before the application object updates its windows.
- `applicationDidUpdate:` (page 70) *delegate method*
  Sent by the default notification center immediately after the application object updates its windows.
- `applicationShouldHandleReopen:hasVisibleWindows:` (page 72) *delegate method*
  Sent by the application to the delegate prior to default behavior to reopen (`rapp`) AppleEvents.

Minimizing Windows

- `miniaturizeAll:` (page 37)
  Miniaturizes all the receiver's windows.

Hiding Windows

- `isHidden` (page 34)
  Returns a Boolean value indicating whether the receiver is hidden.
- `hide:` (page 33)
  Hides all the receiver's windows, and the next application in line is activated.
- `unhide:` (page 58)
  Restores hidden windows to the screen and makes the receiver active.
- `unhideWithoutActivation` (page 59)
  Restores hidden windows without activating their owner (the receiver).

Updating Windows

- `updateWindows` (page 59)
  Sends an `update` message to each onscreen window.
- `setWindowsNeedUpdate:` (page 53)
  Sets whether the receiver's windows need updating when the receiver has finished processing the current event.

Managing Window Layers

- `preventWindowOrdering` (page 42)
  Suppresses the usual window ordering in handling the most recent mouse-down event.
- `arrangeInFront:` (page 25)
  Arranges windows listed in the Window menu in front of all other windows.

Accessing the Main Menu

- `mainMenu` (page 35)
  Returns the receiver's main menu.
- `setMainMenu:` (page 51)
  Makes the given menu the receiver's main menu.

Managing the Window Menu

- `windowsMenu` (page 62)
  Returns the Window menu of the application.
- `setWindowsMenu:` (page 53)
  Makes the given menu the receiver's Window menu.
- `addWindowsItem:title:filename:` (page 24)
  Adds an item to the Window menu for a given window.
- `changeWindowsItem:title:filename:` (page 27)
  Changes the item for a given window in the Window menu to a given string.
- `removeWindowsItem:` (page 43)
  Removes the Window menu item for a given window.
- `updateWindowsItem:` (page 60)
  Updates the Window menu item for a given window to reflect the edited status of that window.

Managing the Dock Menu

- `applicationDockMenu:` (page 71) *delegate method*
  Allows the delegate to supply a dock menu for the application dynamically.

Accessing the Dock Tile

- `dockTile` (page 31)
  Returns the application's Dock tile.

Managing the Services Menu

- `registerServicesMenuSendTypes:returnTypes:` (page 42)
  Registers the pasteboard types the receiver can send and receive in response to service requests.
- `servicesMenu` (page 50)
  Returns the Services menu.
- `setServicesMenu:` (page 52)
  Makes a given menu the receiver's Services menu.

Providing Services

- `validRequestorForSendType:returnType:` (page 60)
  Indicates whether the receiver can send and receive the specified pasteboard types.
- `servicesProvider` (page 50)
  Returns the object that provides the services the receiver advertises in the Services menu of other applications.
- `setServicesProvider:` (page 52)
  Registers a given object as the service provider.

Managing Panels

- `orderFrontColorPanel:` (page 40)
  Brings up the color panel, an instance of `NSColorPanel`.
- `orderFrontStandardAboutPanel:` (page 40)
  Displays a standard About window.
- `orderFrontStandardAboutPanelWithOptions:` (page 41)
  Displays a standard About window with information from a given options dictionary.
- `orderFrontCharacterPalette:` (page 39)
  Opens the character palette.
- `runPageLayout:` (page 48)
  Displays the receiver's page layout panel, an instance of `NSPageLayout`.

Displaying Help

- `showHelp:` (page 53)
  If your project is properly registered, and the necessary keys have been set in the property list, this method launches Help Viewer and displays the first page of your application's help book.
- `activateContextHelpMode:` (page 22)
  Places the receiver in context-sensitive help mode.

Displaying Errors

- `application:willPresentError:` (page 67) *delegate method*
  Sent to the delegate before the specified application presents an error message to the user.

Managing Threads

+ `detachDrawingThread:toTarget:withObject:` (page 21)
  Creates and executes a new thread based on the specified target and selector.

Posting Actions

- `tryToPerform:with:` (page 58)
  Dispatches an action message to the specified target.
- `sendAction:to:from:` (page 48)
  Sends the given action message to the given target.
- `targetForAction:` (page 55)
  Returns the object that receives the action message specified by the given selector
- `targetForAction:to:from:` (page 56)
  Finds an object that can receive the message specified by the given selector.

Drawing Windows

- `context` (page 28)
  Returns the receiver's display context.

Logging Exceptions

- `reportException:` (page 44)
  Logs a given exception by calling `NSLog()`.

Scripting

- `orderedDocuments` (page 38)
  Returns an array of document objects arranged according to the front-to-back ordering of their associated windows.
- `orderedWindows` (page 39)
  Returns an array of window objects arranged according to their front-to-back ordering on the screen.
- `application:delegateHandlesKey:` (page 62) *delegate method*
  Sent by Cocoa's built-in scripting support during execution of `get` or `set` script commands to find out if the delegate can handle operations on the specified key-value key.

Managing User Attention Requests

- `requestUserAttention:` (page 45)
  Starts a user attention request.
- `cancelUserAttentionRequest:` (page 27)
  Cancels a previous user attention request.
- `replyToOpenOrPrint:` (page 44)
  Handles errors that might occur when the user attempts to open or print files.

Managing the Screen

- `applicationDidChangeScreenParameters:` (page 68) *delegate method*
  Sent by the default notification center when the configuration of the displays attached to the computer is changed (either programmatically or when the user changes settings in the Displays control panel).

Opening Files

- `application:openFile:` (page 63) *delegate method*
  Tells the delegate to open a single file.
- `application:openFileWithoutUI:` (page 64) *delegate method*
  Tells the delegate to open a file programmatically.
- `application:openTempFile:` (page 65) *delegate method*
  Tells the delegate to open a temporary file.
- `application:openFiles:` (page 64) *delegate method*
  Tells the delegate to open multiple files.
- `applicationOpenUntitledFile:` (page 71) *delegate method*
  Tells the delegate to open an untitled file.
- `applicationShouldOpenUntitledFile:` (page 73) *delegate method*
  Invoked immediately before opening an untitled file.

Printing

- `application:printFile:` (page 66) *delegate method*
  Sent when the user starts up the application on the command line with the `-NSPrint` option.
- `application:printFiles:withSettings:showPrintPanels:` (page 66) *delegate method*
  Prints a group of files.

Deprecated

- `runModalForWindow:relativeToWindow:` (page 47)
  (Deprecated. Use `beginSheet:modalForWindow:modalDelegate:didEndSelector:contextInfo:` (page 26) instead.)

- `beginModalSessionForWindow:relativeToWindow:` (page 26)
  (Deprecated. Use `beginSheet:modalForWindow:modalDelegate:didEndSelector:contextInfo:` (page 26) instead.)
- `application:printFiles:` (page 89) *delegate method* Deprecated in Mac OS X v10.4
  (Deprecated. Use `application:printFiles:withSettings:showPrintPanels:` (page 66) instead.)

Class Methods

detachDrawingThread:toTarget:withObject:

Creates and executes a new thread based on the specified target and selector.

```
+ (void)detachDrawingThread:(SEL)selector toTarget:(id)target withObject:(id)argument
```

Parameters

selector
The selector whose code you want to execute in the new thread.

target
The object that defines the specified selector.

argument
An optional argument you want to pass to the selector.

Discussion
This method is a convenience wrapper for the `detachNewThreadSelector:toTarget:withObject:` method of `NSThread`. This method automatically creates an `NSAutoreleasePool` object for the new thread before invoking *selector*.

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

Declared In
`NSApplication.h`

sharedApplication

Returns the application instance, creating it if it doesn't exist yet.

```
+ (NSApplication *)sharedApplication
```

Return Value
The shared application object.

Discussion
This method also makes a connection to the window server and completes other initialization. Your program should invoke this method as one of the first statements in `main()`; this invoking is done for you if you create your application with Xcode. To retrieve the `NSApplication` instance after it has been created, use the global variable `NSApp` or invoke this method.

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

See Also
- `run` (page 45)
- `terminate:` (page 57)

Related Sample Code
CoreRecipes
ImageClient
NumberInput_IMKit_Sample
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSApplication.h`

Instance Methods

abortModal

Aborts the event loop started by `runModalForWindow:` (page 46) or `runModalSession:` (page 47).

```
- (void)abortModal
```

Discussion
When stopped with this method, `runModalForWindow:` and `runModalSession:` return `NSRunAbortedResponse`.

`abortModal` must be used instead of `stopModal` (page 55) or `stopModalWithCode:` (page 55) when you need to stop a modal event loop from anywhere other than a callout from that event loop. In other words, if you want to stop the loop in response to a user's actions within the modal window, use `stopModal`; otherwise, use `abortModal`. For example, use `abortModal` when running in a different thread from the Application Kit's main thread or when responding to an `NSTimer` that you have added to the `NSModalPanelRunLoopMode` mode of the default `NSRunLoop`.

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

See Also
- `endModalSession:` (page 31)

Declared In
`NSApplication.h`

activateContextHelpMode:

Places the receiver in context-sensitive help mode.

```
- (void)activateContextHelpMode:(id)sender
```

Parameters

sender

The object that sent the command.

Discussion

In this mode, the cursor becomes a question mark, and help appears for any user interface item the user clicks.

Most applications don't use this method. Instead, applications enter context-sensitive mode when the user presses the Help key. Applications exit context-sensitive help mode upon the first event after a help window is displayed.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `showHelp:` (page 53)

Declared In

`NSHelpManager.h`

activateIgnoringOtherApps:

Makes the receiver the active application.

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

Parameters

flag

If `NO`, the application is activated only if no other application is currently active. If `YES`, the application activates regardless.

Discussion

The *flag* parameter is normally set to `NO`. When the Finder launches an application, using a value of `NO` for *flag* allows the application to become active if the user waits for it to launch, but the application remains unobtrusive if the user activates another application. Regardless of the setting of *flag*, there may be a time lag before the application activates—you should not assume the application will be active immediately after sending this message.

You rarely need to invoke this method. Under most circumstances, the Application Kit takes care of proper activation. However, you might find this method useful if you implement your own methods for interapplication communication.

You don't need to send this message to make one of the application's `NSWindows` key. When you send a `makeKeyWindow` message to an `NSWindow` object, you ensure that it is the key window when the application is active.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `deactivate` (page 29)
- `isActive` (page 34)

Declared In
NSApplication.h

addWindowsItem:title:filename:

Adds an item to the Window menu for a given window.

```
- (void)addWindowsItem:(NSWindow *)aWindow title:(NSString *)aString
    filename:(BOOL)isFilename
```

Parameters

aWindow

The window being added to the menu. If this window object already exists in the Window menu, this method has no effect.

aString

The string to display for the window's menu item. How the string is interpreted is dependent on the value in the *isFilename* parameter.

isFilename

If NO, *aString* appears literally in the menu; otherwise, *aString* is assumed to be a converted pathname with the name of the file preceding the path (the way the NSWindow method setTitleWithRepresentedFilename: shows a title)

Discussion
You rarely need to invoke this method directly because Cocoa places an item in the Window menu automatically whenever you set the title of an NSWindow object.

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

See Also
- changeWindowsItem:title:filename: (page 27)
- setTitle: (NSWindow)

Related Sample Code
QTAudioExtractionPanel

Declared In
NSApplication.h

applicationIconImage

Returns the image used for the receiver's icon.

```
- (NSImage *)applicationIconImage
```

Return Value
An image containing the application's icon.

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

See Also
- `setApplicationIconImage:` (page 50)

Declared In
`NSApplication.h`

arrangeInFront:

Arranges windows listed in the Window menu in front of all other windows.

```
- (void)arrangeInFront:(id)sender
```

Parameters

sender
The object that sent the command.

Discussion
Windows associated with the application but not listed in the Window menu are not ordered to the front.

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

See Also
- `addWindowsItem:title:filename:` (page 24)
- `removeWindowsItem:` (page 43)
- `makeKeyAndOrderFront:` (`NSWindow`)

Declared In
`NSApplication.h`

beginModalSessionForWindow:

Sets up a modal session with the given window and returns an `NSModalSession` structure representing the session.

```
- (NSModalSession)beginModalSessionForWindow:(NSWindow *)aWindow
```

Parameters

aWindow
The window for the session.

Return Value
The `NSModalSession` structure that represents the session.

Discussion
In a modal session, the application receives mouse events only if they occur in *aWindow*. The window is made key, and if not already visible is placed onscreen using the `NSWindow` method `center`.

The `beginModalSessionForWindow:` method only sets up the modal session. To actually run the session, use `runModalSession:` (page 47). `beginModalSessionForWindow:` should be balanced by `endModalSession:` (page 31). Make sure these two messages are sent within the same exception-handling scope. That is, if you send `beginModalSessionForWindow:` inside an `NS_DURING` construct, you must send `endModalSession:` before `NS_ENDHANDLER`.

If an exception is raised, `beginModalSessionForWindow:` arranges for proper cleanup. Do not use `NS_DURING` constructs to send an `endModalSession:` message in the event of an exception.

A loop using these methods is similar to a modal event loop run with `runModalForWindow:` (page 46), except the application can continue processing between method invocations.

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

Declared In
`NSApplication.h`

beginModalSessionForWindow:relativeToWindow:

(**Deprecated.** Use
`beginSheet:modalForWindow:modalDelegate:didEndSelector:contextInfo:` (page 26) instead.)

```
- (NSModalSession)beginModalSessionForWindow:(NSWindow *)theWindow
    relativeToWindow:(NSWindow *)docWindow
```

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

Declared In
`NSApplication.h`

beginSheet:modalForWindow:modalDelegate:didEndSelector:contextInfo:

Starts a document modal session.

```
- (void)beginSheet:(NSWindow *)sheet modalForWindow:(NSWindow *)docWindow
    modalDelegate:(id)modalDelegate didEndSelector:(SEL)didEndSelector
    contextInfo:(void *)contextInfo
```

Parameters

sheet
The window object representing the sheet you want to display.

docWindow
The window object to which you want to attach the sheet.

modalDelegate
The delegate object that defines your *didEndSelector* method. If `nil`, the method in *didEndSelector* is not called.

didEndSelector
An optional method to call when the sheet's modal session has ended. This method must be defined on the object in the *modalDelegate* parameter and have the following signature:

```
- (void)sheetDidEnd:(NSWindow *)sheet returnCode:(NSInteger)returnCode
contextInfo:(void *)contextInfo;
```

contextInfo
A pointer to the context info you want passed to the *didEndSelector* method when the sheet's modal session ends.

Discussion
This method runs the modal event loop for the specified sheet synchronously. It displays the sheet, makes it key, starts the run loop, and processes events for it. While the application is in the run loop, it does not respond to any other events (including mouse, keyboard, or window-close events) unless they are associated with the sheet. It also does not perform any tasks (such as firing timers) that are not associated with the modal run loop. In other words, this method consumes only enough CPU time to process events and dispatch them to the action methods associated with the modal window.

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

See Also
- `endSheet:` (page 31)
- `endSheet:returnCode:` (page 32)

Related Sample Code
IdentitySample
ImageClient
QTSSConnectionMonitor
QTSSInspector
WhackedTV

Declared In
`NSApplication.h`

cancelUserAttentionRequest:

Cancels a previous user attention request.

```
- (void)cancelUserAttentionRequest:(NSInteger)request
```

Parameters
request
The request identifier returned by the `requestUserAttention:` method.

Discussion
A request is also canceled automatically by user activation of the application.

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

See Also
- `requestUserAttention:` (page 45)

Declared In
`NSApplication.h`

changeWindowsItem:title:filename:

Changes the item for a given window in the Window menu to a given string.

```
- (void)changeWindowsItem:(NSWindow *)aWindow title:(NSString *)aString
    filename:(BOOL)isFilename
```

Parameters

aWindow

The window whose title you want to change in the Window menu. If *aWindow* is not in the Window menu, this method adds it.

aString

The string to display for the window's menu item. How the string is interpreted is dependent on the value in the *isFilename* parameter.

isFilename

If NO, *aString* appears literally in the menu; otherwise, *aString* is assumed to be a converted pathname with the name of the file preceding the path (the way the NSWindow method setTitleWithRepresentedFilename: shows a title)

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

See Also
- addWindowsItem:title:filename: (page 24)
- removeWindowsItem: (page 43)
- setTitle: (NSWindow)

Declared In
NSApplication.h

context

Returns the receiver's display context.

```
- (NSGraphicsContext *)context
```

Return Value
The current display context for the application.

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

Declared In
NSApplication.h

currentEvent

Returns the current event, the last event the receiver retrieved from the event queue.

```
- (NSEvent *)currentEvent
```

Return Value
The last event object retrieved by the application.

Discussion
`NSApp` receives events and forwards them to the affected `NSWindow` objects, which then distribute them to the objects in its view hierarchy.

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

See Also
- `discardEventsMatchingMask:beforeEvent:` (page 30)
- `postEvent:atStart:` (page 41)
- `sendEvent:` (page 49)

Related Sample Code
Clock Control

Declared In
`NSApplication.h`

deactivate

Deactivates the receiver.

```
- (void)deactivate
```

Discussion
Normally, you shouldn't invoke this method—the Application Kit is responsible for proper deactivation.

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

See Also
- `activateIgnoringOtherApps:` (page 23)

Declared In
`NSApplication.h`

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Return Value
The application delegate object.

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

See Also
- `setDelegate:` (page 51)

Declared In
`NSApplication.h`

discardEventsMatchingMask:beforeEvent:

Removes all events matching the given mask and generated before the specified event.

```
- (void)discardEventsMatchingMask:(NSUInteger)mask beforeEvent:(NSEvent *)lastEvent
```

Parameters

mask

Contains one or more flags indicating the types of events to discard. The constants section of the `NSEvent` class defines the constants you can add together to create this mask. The discussion section also lists some of the constants that are typically used.

lastEvent

A marker event that you use to indicate which events should be discarded. Events that occurred before this event are discarded but those that occurred after it are not.

Discussion

Use this method to ignore any events that occurred before a specific event. For example, suppose your application has a tracking loop that you exit when the user releases the mouse button. You could use this method, specifying `NSAnyEventMask` as the mask argument and the ending mouse-up event as the *lastEvent* argument, to discard all events that occurred while you were tracking mouse movements in your loop. Passing the mouse-up event as *lastEvent* ensures that any events that might have occurred after the mouse-up event (that is, that appear in the queue after the mouse-up event) are not discarded.

Note: Typically, you send this message to an `NSWindow` object, rather than to the application object. Discarding events for a window clears out all of the events for that window only, leaving events for other windows in place.

For the *mask* parameter, you can add together event type constants such as the following:

```
NSLeftMouseDownMask
NSLeftMouseUpMask
NSRightMouseDownMask
NSRightMouseUpMask
NSMouseMovedMask
NSLeftMouseDraggedMask
NSRightMouseDraggedMask
NSMouseEnteredMask
NSMouseExitedMask
NSKeyDownMask
NSKeyUpMask
NSFlagsChangedMask
NSPeriodicMask
NSCursorUpdateMask
NSAnyEventMask
```

This method can also be called in subthreads. Events posted in subthreads bubble up in the main thread event queue.

Availability

Available in Mac OS X v10.0 and later.

See Also
- `nextEventMatchingMask:untilDate:inMode:dequeue:` (page 37)

Declared In
`NSApplication.h`

dockTile

Returns the application's Dock tile.

```
- (NSDockTile *)dockTile;
```

Return Value
The application's Dock tile.

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

Declared In
`NSApplication.h`

endModalSession:

Finishes a modal session.

```
- (void)endModalSession:(NSModalSession)session
```

Parameters
session
A modal session structure returned by a previous invocation of `beginModalSessionForWindow:`.

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

See Also
- `beginModalSessionForWindow:` (page 25)
- `runModalSession:` (page 47)

Declared In
`NSApplication.h`

endSheet:

Ends a document modal session by specifying the sheet window.

```
- (void)endSheet:(NSWindow *)sheet
```

Parameters
sheet
The sheet whose modal session you want to end.

Discussion
This method ends the modal session with the return code `NSRunStoppedResponse`.

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

See Also
- `beginSheet:modalForWindow:modalDelegate:didEndSelector:contextInfo:` (page 26)
- `endSheet:returnCode:` (page 32)

Related Sample Code
QTSSConnectionMonitor
QTSSInspector
WhackedTV

Declared In
`NSApplication.h`

endSheet:returnCode:

Ends a document modal session by specifying the sheet window.

```
- (void)endSheet:(NSWindow *)sheet returnCode:(NSInteger)returnCode
```

Parameters

sheet
The sheet whose modal session you want to end.

returnCode
The return code to send to the delegate. You can use one of the return codes defined in "Return values for modal operations" (page 77) or a custom value that you define.

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

See Also
- `beginSheet:modalForWindow:modalDelegate:didEndSelector:contextInfo:` (page 26)
- `endSheet:` (page 31)

Related Sample Code
IdentitySample
ImageClient

Declared In
`NSApplication.h`

finishLaunching

Activates the receiver, opens any files specified by the `NSOpen` user default, and unhighlights the application's icon.

```
- (void)finishLaunching
```

Discussion
The `run` (page 45) method invokes this method before it starts the event loop. When this method begins, it posts an `NSApplicationWillFinishLaunchingNotification` (page 86) to the default notification center. If you override `finishLaunching` (page 32), the subclass method should invoke the superclass method.

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

See Also
- `applicationWillFinishLaunching:` (page 75)
- `applicationDidFinishLaunching:` (page 69)

Declared In
`NSApplication.h`

hide:

Hides all the receiver's windows, and the next application in line is activated.

```
- (void)hide:(id)sender
```

Parameters

sender
The object that sent the command.

Discussion
This method is usually invoked when the user chooses Hide in the application's main menu. When this method begins, it posts an `NSApplicationWillHideNotification` (page 86) to the default notification center. When it completes successfully, it posts an `NSApplicationDidHideNotification` (page 85).

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

See Also
- `miniaturizeAll:` (page 37)
- `unhide:` (page 58)
- `unhideWithoutActivation` (page 59)
- `applicationDidHide:` (page 69)
- `applicationWillHide:` (page 75)

Declared In
`NSApplication.h`

hideOtherApplications:

Hides all applications, except the receiver.

```
- (void)hideOtherApplications:(id)sender
```

Parameters

sender

The object that sent this message.

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

Declared In
NSApplication.h

isActive

Returns a Boolean value indicating whether this is the active application.

```
- (BOOL)isActive
```

Return Value
YES if this is the active application; NO otherwise.

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

See Also
- activateIgnoringOtherApps: (page 23)
- deactivate (page 29)

Declared In
NSApplication.h

isHidden

Returns a Boolean value indicating whether the receiver is hidden.

```
- (BOOL)isHidden
```

Return Value
YES if the receiver is hidden, NO otherwise.

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

See Also
- hide: (page 33)
- unhide: (page 58)
- unhideWithoutActivation (page 59)

Declared In
NSApplication.h

isRunning

Returns a Boolean value indicating whether the main event loop is running.

```
- (BOOL)isRunning
```

Return Value
`YES` if the main event loop is running; `NO` otherwise.

Discussion
`NO` means the `stop:` (page 54) method was invoked.

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

See Also
- `run` (page 45)
- `terminate:` (page 57)

Declared In
`NSApplication.h`

keyWindow

Returns the window that currently receives keyboard events.

```
- (NSWindow *)keyWindow
```

Return Value
The window object currently receiving keyboard events or `nil` if there is no key window.

Discussion
This method might return `nil` if the application's nib file hasn't finished loading yet or if the receiver is not active.

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

See Also
- `mainWindow` (page 36)
- `isKeyWindow` (NSWindow)

Declared In
`NSApplication.h`

mainMenu

Returns the receiver's main menu.

```
- (NSMenu *)mainMenu
```

Return Value
The menu object representing the application's menu bar.

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

See Also
- `setMainMenu:` (page 51)

Declared In
`NSApplication.h`

mainWindow

Returns the main window.

```
- (NSWindow *)mainWindow
```

Return Value
The application's main window or `nil` if there is no main window.

Discussion
This method might return `nil` if the application's nib file hasn't finished loading, if the receiver is not active, or if the application is hidden.

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

See Also
- `keyWindow` (page 35)
- `isMainWindow` (NSWindow)

Declared In
`NSApplication.h`

makeWindowsPerform:inOrder:

Sends the specified message to each of the application's window objects until one returns a non-`nil` value.

```
- (NSWindow *)makeWindowsPerform:(SEL)aSelector inOrder:(BOOL)flag
```

Parameters

aSelector

The selector to perform on each window. This method must not take any arguments and must return a value whose type that can be compared to `nil`.

flag

If `YES`, the *aSelector* message is sent to each of the window server's onscreen windows, going in z-order, until one returns a non-`nil` value. A minimized window is not considered to be onscreen for this check. If `NO`, the message is sent to all windows in `NSApp`'s window list, regardless of whether or not they are onscreen. This order is unspecified.

Return Value
The window that returned a non-`nil` value or `nil` if all windows returned nil from *aSelector*.

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

See Also
- `sendAction:to:from:` (page 48)

- `tryToPerform:with:` (page 58)
- `windows` (page 61)

Declared In
`NSApplication.h`

miniaturizeAll:

Miniaturizes all the receiver's windows.

```
- (void)miniaturizeAll:(id)sender
```

Parameters

sender
The object that sent the command.

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

See Also
- `hide:` (page 33)

Declared In
`NSApplication.h`

modalWindow

Returns the modal window that the receiver is displaying.

```
- (NSWindow *)modalWindow
```

Return Value
The modal window being displayed or `nil` if no modal window is being displayed.

Discussion
This method returns the current standalone modal window. It does not return sheets that are attached to other windows. If you need to retrieve a sheet window, use the `attachedSheet` method of `NSWindow`.

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

Declared In
`NSApplication.h`

nextEventMatchingMask:untilDate:inMode:dequeue:

Returns the next event matching a given mask, or `nil` if no such event is found before a specified expiration date.

```
- (NSEvent *)nextEventMatchingMask:(NSUInteger)mask untilDate:(NSDate *)expiration
    inMode:(NSString *)mode dequeue:(BOOL)flag
```

Parameters

mask

Contains one or more flags indicating the types of events to return. The constants section of the `NSEvent` class defines the constants you can add together to create this mask. The `discardEventsMatchingMask:beforeEvent:` (page 30) method also lists several of these constants.

expiration

The expiration date for the current event request. Specifying nil for this parameter is equivalent to returning a date object using the `distantPast` method.

mode

The run loop mode in which to run while looking for events. The mode you specify also determines which timers and run-loop observers may fire while the application waits for the event.

flag

Specify `YES` if you want the event removed from the queue.

Return Value

The event object whose type matches one of the event types specified by the *mask* parameter.

Discussion

You can use this method to short circuit normal event dispatching and get your own events. For example, you may want to do this in response to a mouse-down event in order to track the mouse while its button is down. (In such an example, you would pass the appropriate event types for mouse-dragged and mouse-up events to the *mask* parameter and specify the `NSEventTrackingRunLoopMode` run loop mode.) Events that do not match one of the specified event types are left in the queue.

You can specify one of the run loop modes defined by the Application Kit or a custom run loop mode used specifically by your application. Application Kit defines the following run-loop modes:

```
NSDefaultRunLoopMode
NSEventTrackingRunLoopMode
NSModalPanelRunLoopMode
NSConnectionReplyMode
```

Availability

Available in Mac OS X v10.0 and later.

See Also

- `postEvent:atStart:` (page 41)
- `run` (page 45)
- `runModalForWindow:` (page 46)

Declared In

`NSApplication.h`

orderedDocuments

Returns an array of document objects arranged according to the front-to-back ordering of their associated windows.

```
- (NSArray *)orderedDocuments
```

Return Value
An array of `NSDocument` objects, where the position of a document is based on the front-to-back ordering of its associated window.

Discussion
This method is called during script command evaluation—for example, while finding the document in the script statement `the third rectangle in the first document`. For information on how your application can return its own array of ordered documents, see `application:delegateHandlesKey:` (page 62).

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

See Also
`- orderedWindows` (page 39)

Declared In
`NSApplicationScripting.h`

orderedWindows

Returns an array of window objects arranged according to their front-to-back ordering on the screen.

```
- (NSArray *)orderedWindows
```

Return Value
An array of `NSWindow` objects, where the position of each window in the array corresponds to the front-to-back ordering of the windows on the screen.

Discussion
Only windows that are typically scriptable are included in the returned array. For example, panels are not included.

This method is called during script command evaluation—for example, while finding the window in the script statement `close the second window`. For information on how your application can return its own array of ordered windows, see `application:delegateHandlesKey:` (page 62).

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

See Also
`- orderedDocuments` (page 38)

Declared In
`NSApplicationScripting.h`

orderFrontCharacterPalette:

Opens the character palette.

```
- (void)orderFrontCharacterPalette:(id)sender
```

Parameters

sender

The object that sent the command.

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

Declared In
NSApplication.h

orderFrontColorPanel:

Brings up the color panel, an instance of `NSColorPanel`.

```
- (void)orderFrontColorPanel:(id)sender
```

Parameters

sender

The object that sent the command.

Discussion
If the `NSColorPanel` object does not exist yet, this method creates one. This method is typically invoked when the user chooses Colors from a menu.

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

Declared In
NSColorPanel.h

orderFrontStandardAboutPanel:

Displays a standard About window.

```
- (void)orderFrontStandardAboutPanel:(id)sender
```

Parameters

sender

The object that sent the command.

Discussion
This method calls `orderFrontStandardAboutPanelWithOptions:` (page 41) with a `nil` argument. See `orderFrontStandardAboutPanelWithOptions:` for a description of what's displayed.

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

Related Sample Code
MenuItemView

Declared In
NSApplication.h

orderFrontStandardAboutPanelWithOptions:

Displays a standard About window with information from a given options dictionary.

```
- (void)orderFrontStandardAboutPanelWithOptions:(NSDictionary *)optionsDictionary
```

Parameters

optionsDictionary

A dictionary whose keys define the contents of the About window. See the discussion for a description of the available keys.

Discussion

The following strings are keys that can occur in *optionsDictionary*:

- `@"Credits"`: An NSAttributedString displayed in the info area of the panel. If not specified, this method then looks for a file named "`Credits.html`", "`Credits.rtf`", and "`Credits.rtfd`", in that order, in the bundle returned by the `NSBundle` class method `mainBundle`. The first file found is used. If none is found, the info area is left blank.
- `@"ApplicationName"`: An `NSString` object displayed as the application's name. If not specified, this method then uses the value of `CFBundleName` (localizable). If neither is found, this method uses `[[NSProcessInfo processInfo] processName]`.
- `@"ApplicationIcon"`: An `NSImage` object displayed as the application's icon. If not specified, this method then looks for an image named "`NSApplicationIcon`", using `[NSImage imageNamed:@"NSApplicationIcon"]`. If neither is available, this method uses the generic application icon.
- `@"Version"`: An `NSString` object with the build version number of the application ("58.4"), displayed as "(v58.4)". If not specified, obtain from the `CFBundleVersion` key in `infoDictionary`; if not specified, leave blank (the "(v)" is not displayed).
- `@"Copyright"`: An `NSString` object with a line of copyright information. If not specified, this method then looks for the value of `NSHumanReadableCopyright` in the localized version `infoDictionary`. If neither is available, this method leaves the space blank.
- `@"ApplicationVersion"`: An `NSString` object with the application version ("Mac OS X", "3", "WebObjects 4.5", "AppleWorks 6",...). If not specified, obtain from the `CFBundleShortVersionString` key in `infoDictionary`. If neither is available, the build version, if available, is printed alone, as "Version x.x".

Availability

Available in Mac OS X v10.0 and later.

See Also

`- orderFrontStandardAboutPanel:` (page 40)

Declared In

`NSApplication.h`

postEvent:atStart:

Adds a given event to the receiver's event queue.

```
- (void)postEvent:(NSEvent *)anEvent atStart:(BOOL)flag
```

Parameters

anEvent

The event object to post to the queue.

flag

Specify `YES` to add the event to the front of the queue; otherwise, specify `NO` to add the event to the back of the queue.

Discussion

This method can also be called in subthreads. Events posted in subthreads bubble up in the main thread event queue.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `currentEvent` (page 28)
- `sendEvent:` (page 49)

Declared In

`NSApplication.h`

preventWindowOrdering

Suppresses the usual window ordering in handling the most recent mouse-down event.

```
- (void)preventWindowOrdering
```

Discussion

This method is only useful for mouse-down events when you want to prevent the window that receives the event from being ordered to the front.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSApplication.h`

registerServicesMenuSendTypes:returnTypes:

Registers the pasteboard types the receiver can send and receive in response to service requests.

```
- (void)registerServicesMenuSendTypes:(NSArray *)sendTypes returnTypes:(NSArray
    *)returnTypes
```

Parameters

sendTypes

An array of `NSString` objects, each of which corresponds to a particular pasteboard type that the application can send.

returnTypes

An array of `NSString` objects, each of which corresponds to a particular pasteboard type that the application can receive.

Discussion
If the receiver has a Services menu, a menu item is added for each service provider that can accept one of the specified *sendTypes* or return one of the specified *returnTypes*. You should typically invoke this method at application startup time or when an object that can use services is created. You can invoke it more than once—its purpose is to ensure there is a menu item for every service the application can use. The event-handling mechanism will dynamically enable the individual items to indicate which services are currently appropriate. All the `NSResponder` objects in your application (typically `NSView` objects) should register every possible type they can send and receive by sending this message to `NSApp`.

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

See Also
- `validRequestorForSendType:returnType:` (page 60)
- `readSelectionFromPasteboard:` (NSServicesRequests protocol)
- `writeSelectionToPasteboard:types:` (NSServicesRequests protocol)

Declared In
`NSApplication.h`

removeWindowsItem:

Removes the Window menu item for a given window.

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

Parameters
aWindow
The window whose menu item is to be removed.

Discussion
This method doesn't prevent the item from being automatically added again. Use the `setExcludedFromWindowsMenu:` method of `NSWindow` if you want the item to remain excluded from the Window menu.

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

See Also
- `addWindowsItem:title:filename:` (page 24)
- `changeWindowsItem:title:filename:` (page 27)

Declared In
`NSApplication.h`

replyToApplicationShouldTerminate:

Responds to `NSTerminateLater` once the application knows whether it can terminate.

```
- (void)replyToApplicationShouldTerminate:(BOOL)shouldTerminate
```

Parameters

shouldTerminate

Specify YES if you want the application to terminate; otherwise, specify NO.

Discussion

If your application delegate returns NSTerminateLater from its applicationShouldTerminate: (page 73) method, your code must subsequently call this method to let the NSApplication object know whether it can actually terminate itself.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ExtractMovieAudioToAIFF

QTExtractAndConvertToAIFF

QTExtractAndConvertToMovieFile

Declared In

NSApplication.h

replyToOpenOrPrint:

Handles errors that might occur when the user attempts to open or print files.

```
- (void)replyToOpenOrPrint:(NSApplicationDelegateReply)reply
```

Parameters

reply

The error that occurred. For a list of possible values, see "Constants" (page 77).

Discussion

Delegates should invoke this method if an error is encountered in the application:openFiles: (page 64) or application:printFiles: (page 89) delegate methods.

Availability

Available in Mac OS X v10.3 and later.

Declared In

NSApplication.h

reportException:

Logs a given exception by calling NSLog().

```
- (void)reportException:(NSException *)anException
```

Parameters

anException

The exception whose contents you want to write to the log file.

Discussion

This method does not raise *anException*. Use it inside of an exception handler to record that the exception occurred.

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

See Also
`NSSetUncaughtExceptionHandler` (Foundation Functions)

Declared In
`NSApplication.h`

requestUserAttention:

Starts a user attention request.

```
- (NSInteger)requestUserAttention:(NSRequestUserAttentionType)requestType
```

Parameters

requestType
The severity of the request. For a list of possible values, see "Constants" (page 77).

Return Value
The identifier for the request. You can use this value to cancel the request later using the `cancelUserAttentionRequest:` method.

Discussion
Activating the application cancels the user attention request. A spoken notification will occur if spoken notifications are enabled. Sending `requestUserAttention:` to an application that is already active has no effect.

If the inactive application presents a modal panel, this method will be invoked with `NSCriticalRequest` automatically. The modal panel is not brought to the front for an inactive application.

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

See Also
`- cancelUserAttentionRequest:` (page 27)

Declared In
`NSApplication.h`

run

Starts the main event loop.

```
- (void)run
```

Discussion
The loop continues until a `stop:` (page 54) or `terminate:` (page 57) message is received. Upon each iteration through the loop, the next available event from the window server is stored and then dispatched by sending it to `NSApp` using `sendEvent:` (page 49).

After creating the `NSApplication` object, the `main` function should load your application's main nib file and then start the event loop by sending the `NSApplication` object a `run` message. If you create an Cocoa application project in Xcode, this `main` function is implemented for you.

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

See Also
- `runModalForWindow:` (page 46)
- `runModalSession:` (page 47)
- `applicationDidFinishLaunching:` (page 69)

Related Sample Code
NumberInput_IMKit_Sample

Declared In
`NSApplication.h`

runModalForWindow:

Starts a modal event loop for a given window.

```
- (NSInteger)runModalForWindow:(NSWindow *)aWindow
```

Parameters

aWindow

The window to be displayed modally. If it is not already visible, the window is centered on the screen using the value in its `center`method and made visible and key. If it is already visible, it is simply made key.

Return Value
An integer indicating the reason that this method returned. See the discussion for a description of possible return values.

Discussion
This method runs a modal event loop for the specified window synchronously. It displays the specified window, makes it key, starts the run loop, and processes events for that window. (You do not need to show the window yourself.) While the application is in that loop, it does not respond to any other events (including mouse, keyboard, or window-close events) unless they are associated with the window. It also does not perform any tasks (such as firing timers) that are not associated with the modal run loop. In other words, this method consumes only enough CPU time to process events and dispatch them to the action methods associated with the modal window.

You can exit the modal loop by calling the `stopModal`, `stopModalWithCode:`, or `abortModal` methods from your modal window code. If you use the `stopModalWithCode:` method to stop the modal event loop, this method returns the argument passed to `stopModalWithCode:`. If you use `stopModal` instead, this method returns the constant `NSRunStoppedResponse`. If you use `abortModal`, this method returns the constant `NSRunAbortedResponse`.

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

See Also
- `run` (page 45)

- runModalSession: (page 47)

Related Sample Code
WhackedTV

Declared In
NSApplication.h

runModalForWindow:relativeToWindow:

(**Deprecated.** Use
beginSheet:modalForWindow:modalDelegate:didEndSelector:contextInfo: (page 26) instead.)

```
- (NSInteger)runModalForWindow:(NSWindow *)theWindow relativeToWindow:(NSWindow
    *)docWindow
```

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

Declared In
NSApplication.h

runModalSession:

Runs a given modal session, as defined in a previous invocation of `beginModalSessionForWindow:`.

```
- (NSInteger)runModalSession:(NSModalSession)session
```

Parameters

session
: The modal session structure returned by the `beginModalSessionForWindow:` method for the window to be displayed.

Return Value
An integer indicating the reason that this method returned. See the discussion for a description of possible return values.

Discussion
A loop that uses this method is similar in some ways to a modal event loop run with `runModalForWindow:`, except with this method your code can do some additional work between method invocations. When you invoke this method, events for the `NSWindow` object of this session are dispatched as normal. This method returns when there are no more events. You must invoke this method frequently enough in your loop that the window remains responsive to events. However, you should not invoke this method in a tight loop because it returns immediately if there are no events, and consequently you could end up polling for events rather than blocking.

Typically, you use this method in situations where you want to do some additional processing on the current thread while the modal loop runs. For example, while processing a large data set, you might want to use a modal dialog to display progress and give the user a chance to cancel the operation. If you want to display a modal dialog and do not need to do any additional work in parallel, use `runModalForWindow:` instead. When there are no pending events, that method waits idly instead of consuming CPU time.

The following code shows a sample loop you can use in your code:

```
NSModalSession session = [NSApp beginModalSessionForWindow:theWindow];
for (;;) {
    if ([NSApp runModalSession:session] != NSRunContinuesResponse)
        break;
    [self doSomeWork];
}
[NSApp endModalSession:session];
```

If the modal session was not stopped, this method returns `NSRunContinuesResponse`. At this point, your application can do some work before the next invocation of `runModalSession:` (as indicated in the example's `doSomeWork` call). If `stopModal` (page 55) was invoked as the result of event processing, `runModalSession:` returns `NSRunStoppedResponse`. If `stopModalWithCode:` (page 55) was invoked, this method returns the value passed to `stopModalWithCode:`. If `abortModal` (page 22) was invoked, this method returns `NSRunAbortedResponse`.

The window is placed on the screen and made key as a result of the `runModalSession:` message. Do not send a separate `makeKeyAndOrderFront:` message.

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

See Also
- `beginModalSessionForWindow:` (page 25)
- `endModalSession:` (page 31)
- `run` (page 45)
- `runModalForWindow:` (page 46)

Declared In
`NSApplication.h`

runPageLayout:

Displays the receiver's page layout panel, an instance of `NSPageLayout`.

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

Parameters

sender
The object that sent the command.

Discussion
If the `NSPageLayout` instance does not exist, this method creates one. This method is typically invoked when the user chooses Page Setup from the application's FIle menu.

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

Declared In
`NSPageLayout.h`

sendAction:to:from:

Sends the given action message to the given target.

```
- (BOOL)sendAction:(SEL)anAction to:(id)aTarget from:(id)sender
```

Parameters

anAction

The action message you want to send.

aTarget

The target object that defines the specified action message.

sender

The object to pass for the action message's parameter.

Return Value

`YES` if the action was successfully sent; otherwise `NO`. This method also returns `NO` if *anAction* is `nil`.

Discussion

If *aTarget* is `nil`, `NSApp` looks for an object that can respond to the message—that is, an object that implements a method matching *anAction*. It begins with the first responder of the key window. If the first responder can't respond, it tries the first responder's next responder and continues following next responder links up the responder chain. If none of the objects in the key window's responder chain can handle the message, `NSApp` attempts to send the message to the key window's delegate.

If the delegate doesn't respond and the main window is different from the key window, `NSApp` begins again with the first responder in the main window. If objects in the main window can't respond, `NSApp` attempts to send the message to the main window's delegate. If still no object has responded, `NSApp` tries to handle the message itself. If `NSApp` can't respond, it attempts to send the message to its own delegate.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `targetForAction:` (page 55)
- `tryToPerform:with:` (page 58)
- `makeWindowsPerform:inOrder:` (page 36)

Declared In

`NSApplication.h`

sendEvent:

Dispatches an event to other objects.

```
- (void)sendEvent:(NSEvent *)anEvent
```

Parameters

anEvent

The event object to dispatch.

Discussion

You rarely invoke `sendEvent:` directly, although you might want to override this method to perform some action on every event. `sendEvent:` messages are sent from the main event loop (the `run` (page 45) method). `sendEvent:` is the method that dispatches events to the appropriate responders—`NSApp` handles application events, the `NSWindow` object indicated in the event record handles window-related events, and mouse and key events are forwarded to the appropriate `NSWindow` object for further dispatching.

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

See Also
- `currentEvent` (page 28)
- `postEvent:atStart:` (page 41)

Declared In
`NSApplication.h`

servicesMenu

Returns the Services menu.

```
- (NSMenu *)servicesMenu
```

Return Value
The Services menu or `nil` if no Services menu has been created

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

See Also
- `setServicesMenu:` (page 52)

Declared In
`NSApplication.h`

servicesProvider

Returns the object that provides the services the receiver advertises in the Services menu of other applications.

```
- (id)servicesProvider
```

Return Value
The application's service provider object.

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

See Also
- `setServicesProvider:` (page 52)

Declared In
`NSApplication.h`

setApplicationIconImage:

Sets the receiver's icon to the specified image.

```
- (void)setApplicationIconImage:(NSImage *)anImage
```

Parameters

anImage

The image to use as the new application icon.

Discussion

This method sets the icon in the dock application tile. This method scales the image as necessary so that it fits in the dock tile. You can use this method to change your application icon while running. To restore your application's original icon, you pass `nil` to this method.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- applicationIconImage` (page 24)

Declared In

`NSApplication.h`

setDelegate:

Makes the given object the receiver's delegate.

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

Parameters

anObject

The application delegate object.

Discussion

The messages a delegate can expect to receive are listed at the end of this specification. The delegate doesn't need to implement all the methods.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- delegate` (page 29)

Related Sample Code

CocoaDVDPlayer

JavaSplashScreen

PictureSharing

Declared In

`NSApplication.h`

setMainMenu:

Makes the given menu the receiver's main menu.

```
- (void)setMainMenu:(NSMenu *)aMenu
```

Parameters

aMenu

The new menu bar for the application.

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

See Also
- mainMenu (page 35)

Declared In
NSApplication.h

setServicesMenu:

Makes a given menu the receiver's Services menu.

```
- (void)setServicesMenu:(NSMenu *)aMenu
```

Parameters

aMenu

The new Services menu.

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

See Also
- servicesMenu (page 50)

Declared In
NSApplication.h

setServicesProvider:

Registers a given object as the service provider.

```
- (void)setServicesProvider:(id)aProvider
```

Parameters

aProvider

The new service provider object.

Discussion
The service provider is an object that performs all services the application provides to other applications. When another application requests a service from the receiver, it sends the service request to *aProvider*. Service requests can arrive immediately after the service provider is set, so invoke this method only when your application is ready to receive requests.

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

See Also
- servicesProvider (page 50)

Declared In
NSApplication.h

setWindowsMenu:

Makes the given menu the receiver's Window menu.

```
- (void)setWindowsMenu:(NSMenu *)aMenu
```

Parameters

aMenu

The new Window menu for the application.

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

See Also
- windowsMenu (page 62)

Declared In
NSApplication.h

setWindowsNeedUpdate:

Sets whether the receiver's windows need updating when the receiver has finished processing the current event.

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

Parameters

flag

If YES, the receiver's windows are updated after an event is processed.

Discussion
This method is especially useful for making sure menus are updated to reflect changes not initiated by user actions, such as messages received from remote objects.

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

See Also
- updateWindows (page 59)

Declared In
NSApplication.h

showHelp:

If your project is properly registered, and the necessary keys have been set in the property list, this method launches Help Viewer and displays the first page of your application's help book.

```
- (void)showHelp:(id)sender
```

Parameters

sender

The object that sent the command.

Discussion

For information on how to set up your project to take advantage of having Help Viewer display your help book, see Specifying the Comprehensive Help File.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `activateContextHelpMode:` (page 22)

Related Sample Code

HelpHook

Declared In

`NSHelpManager.h`

stop:

Stops the main event loop.

```
- (void)stop:(id)sender
```

Parameters

sender

The object that sent this message.

Discussion

This method notifies the application that you want to exit the current run loop as soon as it finishes processing the current `NSEvent` object. This method does not forcibly exit the current run loop. Instead it sets a flag that the application checks only after it finishes dispatching an actual event object. For example, you could call this method from an action method responding to a button click or from one of the many methods defined by the `NSResponder` class. However, calling this method from a timer or run-loop observer routine would not stop the run loop because they do not result in the p of an `NSEvent` object.

If you call this method from an event handler running in your main run loop, the application object exits out of the `run` method, thereby returning control to the `main()` function. If you call this method from within a modal event loop, it will exit the modal loop instead of the main event loop.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `run` (page 45)
- `runModalForWindow:` (page 46)
- `runModalSession:` (page 47)
- `terminate:` (page 57)

Declared In

`NSApplication.h`

stopModal

Stops a modal event loop.

```
- (void)stopModal
```

Discussion
This method should always be paired with a previous invocation of `runModalForWindow:` (page 46) or `beginModalSessionForWindow:` (page 25). When `runModalForWindow:` (page 46) is stopped with this method, it returns `NSRunStoppedResponse`. This method stops the loop only if it's executed by code responding to an event. If you need to stop a `runModalForWindow:` (page 46) loop outside of one of its event callbacks (for example, a method repeatedly invoked by an `NSTimer` object or a method running in a different thread), use the `abortModal` (page 22) method.

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

See Also
- `runModalSession:` (page 47)
- `stopModalWithCode:` (page 55)

Related Sample Code
WhackedTV

Declared In
`NSApplication.h`

stopModalWithCode:

Stops a modal event loop, allowing you to return a custom result code.

```
- (void)stopModalWithCode:(NSInteger)returnCode
```

Parameters

returnCode
The result code you want returned from the `runModalForWindow:` or `runModalSession:` method. The meaning of this result code is up to you.

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

See Also
- `abortModal` (page 22)
- `runModalForWindow:` (page 46)

Declared In
`NSApplication.h`

targetForAction:

Returns the object that receives the action message specified by the given selector

```
- (id)targetForAction:(SEL)aSelector
```

Parameters

aSelector

The desired action message.

Return Value

The object that would receive the specified action message or `nil` if no target object would receive the message. This method also returns `nil` if *aSelector* is `nil`.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `sendAction:to:from:` (page 48)
- `tryToPerform:with:` (page 58)
- `targetForAction:to:from:` (page 56)

Declared In

`NSApplication.h`

targetForAction:to:from:

Finds an object that can receive the message specified by the given selector.

```
- (id)targetForAction:(SEL)anAction to:(id)aTarget from:(id)sender
```

Parameters

anAction

The desired action message.

aTarget

The first target object to check. Specify `nil` if you want the application to search the responder chain.

sender

The parameter to send to the action message.

Return Value

The object that can accept the specified action message or `nil` if no target object can receive the message. This method also returns `nil` if *anAction* is `nil`.

Discussion

If *aTarget* is not `nil`, *aTarget* is returned. If *aTarget* is `nil`, `NSApp` looks for an object that can respond to the message—that is, an object that implements a method matching *anAction*. The search begins with the first responder of the key window. If the first responder does not handle the message, it tries the first responder's next responder and continues following next responder links up the responder chain. If none of the objects in the key window's responder chain can handle the message, `NSApp` asks the key window's delegate whether it can handle the message.

If the delegate cannot handle the message and the main window is different from the key window, `NSApp` begins searching again with the first responder in the main window. If objects in the main window cannot handle the message, `NSApp` tries the main window's delegate. If it cannot handle the message, `NSApp` asks itself. If `NSApp` doesn't handle the message, it asks the application delegate. If there is no object capable of handling the message, `nil` is returned.

Availability

Available in Mac OS X v10.0 and later.

See Also
- `sendAction:to:from:` (page 48)
- `tryToPerform:with:` (page 58)
- `targetForAction:` (page 55)

Declared In
`NSApplication.h`

terminate:

Terminates the receiver.

```
- (void)terminate:(id)sender
```

Parameters

sender

Typically, this parameter contains the object that initiated the termination request.

Discussion
This method is typically invoked when the user chooses Quit or Exit from the application's menu.

When invoked, this method performs several steps to process the termination request. First, it asks the application's document controller (if one exists) to save any unsaved changes in its documents. During this process, the document controller can cancel termination in response to input from the user. If the document controller does not cancel the operation, this method then calls the delegate's `applicationShouldTerminate:` method. If `applicationShouldTerminate:` returns NSTerminateCancel, the termination process is aborted and control is handed back to the main event loop. If the method returns `NSTerminateLater`, the application runs its run loop in the `NSModalPanelRunLoopMode` mode until the `replyToApplicationShouldTerminate:` method is called with the value `YES` or `NO`. If the `applicationShouldTerminate:` method returns NSTerminateNow, this method posts a `NSApplicationWillTerminateNotification` notification to the default notification center.

Do not bother to put final cleanup code in your application's `main()` function—it will never be executed. If cleanup is necessary, perform that cleanup in the delegate's `applicationWillTerminate:` method.

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

See Also
- `run` (page 45)
- `stop:` (page 54)
- `applicationShouldTerminate:` (page 73)
- `applicationWillTerminate:` (page 76)
- `replyToApplicationShouldTerminate:` (page 43)

`NSApplicationWillTerminateNotification` (page 87)

Related Sample Code
JavaSplashScreen
QTSSInspector
StickiesExample
WhackedTV

Declared In
NSApplication.h

tryToPerform:with:

Dispatches an action message to the specified target.

```
- (BOOL)tryToPerform:(SEL)aSelector with:(id)anObject
```

Parameters

aSelector
The action message you want to dispatch.

anObject
The target object that defines the specified selector.

Return Value
YES if either the receiver or its delegate can accept the specified selector; otherwise, NO. This method also returns NO if *aSelector* is nil.

Discussion
The receiver tries to perform the method *aSelector* using its inherited tryToPerform:with: method of NSResponder. If the receiver doesn't perform *aSelector*, the delegate is given the opportunity to perform it using its inherited performSelector:withObject: method of NSObject.

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

See Also
respondsToSelector: (NSObject protocol)

Declared In
NSApplication.h

unhide:

Restores hidden windows to the screen and makes the receiver active.

```
- (void)unhide:(id)sender
```

Parameters

sender
The object that sent the command.

Discussion
Invokes unhideWithoutActivation (page 59).

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

See Also
- activateIgnoringOtherApps: (page 23)
- hide: (page 33)

Declared In
NSApplication.h

unhideAllApplications:

Unhides all applications, including the receiver.

```
- (void)unhideAllApplications:(id)sender
```

Parameters

sender
The object that sent this message.

Discussion
This action causes each application to order its windows to the front, which could obscure the currently active window in the active application.

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

Declared In
NSApplication.h

unhideWithoutActivation

Restores hidden windows without activating their owner (the receiver).

```
- (void)unhideWithoutActivation
```

Discussion
When this method begins, it posts an `NSApplicationWillUnhideNotification` (page 87) to the default notification center. If it completes successfully, it posts an `NSApplicationDidUnhideNotification` (page 85).

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

See Also
- `activateIgnoringOtherApps:` (page 23)
- `hide:` (page 33)
- `applicationDidUnhide:` (page 70)
- `applicationWillUnhide:` (page 76)

Declared In
NSApplication.h

updateWindows

Sends an `update` message to each onscreen window.

```
- (void)updateWindows
```

Discussion
This method is invoked automatically in the main event loop after each event when running in `NSDefaultRunLoopMode` or `NSModalRunLoopMode`. This method is not invoked automatically when running in `NSEventTrackingRunLoopMode`.

When this method begins, it posts an `NSApplicationWillUpdateNotification` (page 87) to the default notification center. When it successfully completes, it posts an `NSApplicationDidUpdateNotification` (page 85).

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

See Also
- `update` (NSWindow)
- `setWindowsNeedUpdate:` (page 53)
- `applicationDidUpdate:` (page 70)
- `applicationWillUpdate:` (page 77)

Declared In
`NSApplication.h`

updateWindowsItem:

Updates the Window menu item for a given window to reflect the edited status of that window.

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

Parameters
aWindow
The window whose menu item is to be updated.

Discussion
You rarely need to invoke this method because it is invoked automatically when the edit status of an `NSWindow` object is set.

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

See Also
- `changeWindowsItem:title:filename:` (page 27)
- `setDocumentEdited:` (NSWindow)

Declared In
`NSApplication.h`

validRequestorForSendType:returnType:

Indicates whether the receiver can send and receive the specified pasteboard types.

```
- (id)validRequestorForSendType:(NSString *)sendType returnType:(NSString
    *)returnType
```

Parameters

sendType

The pasteboard type the application needs to send.

returnType

The pasteboard type the application needs to receive.

Return Value

The object that can send and receive the specified types or `nil` if the receiver knows of no object that can send and receive data of that type.

Discussion

This message is sent to all responders in a responder chain. `NSApp` is typically the last item in the responder chain, so it usually receives this message only if none of the current responders can send *sendType* data and accept back *returnType* data.

The receiver passes this message on to its delegate if the delegate can respond (and isn't an `NSResponder` object with its own next responder). If the delegate cannot respond or returns `nil`, this method returns `nil`. If the delegate can find an object that can send *sendType* data and accept back *returnType* data, it returns that object.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `registerServicesMenuSendTypes:returnTypes:` (page 42)
- `validRequestorForSendType:returnType:` (NSResponder)
- `readSelectionFromPasteboard:` (NSServicesRequests protocol)
- `writeSelectionToPasteboard:types:` (NSServicesRequests protocol)

Declared In

`NSApplication.h`

windows

Returns an array containing the receiver's window objects.

```
- (NSArray *)windows
```

Return Value

An array of `NSWindow` objects. This array includes both onscreen and offscreen windows.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In

`NSApplication.h`

windowsMenu

Returns the Window menu of the application.

```
- (NSMenu *)windowsMenu
```

Return Value
The window menu or `nil` if such a menu does not exist or has not yet been created.

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

See Also
`- setWindowsMenu:` (page 53)

Declared In
`NSApplication.h`

windowWithWindowNumber:

Returns the window corresponding to the specified window number.

```
- (NSWindow *)windowWithWindowNumber:(NSInteger)windowNum
```

Parameters
windowNum
The unique window number associated with the desired `NSWindow` object.

Return Value
The desired window object or `nil` if the window could not be found.

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

Declared In
`NSApplication.h`

Delegate Methods

application:delegateHandlesKey:

Sent by Cocoa's built-in scripting support during execution of `get` or `set` script commands to find out if the delegate can handle operations on the specified key-value key.

```
- (BOOL)application:(NSApplication *)sender delegateHandlesKey:(NSString *)key
```

Parameters
sender
The application object associated with the delegate.

key
The key to be handled.

Return Value
`YES` if your delegate handles the key or `NO` if it does not.

Discussion
The method should return `YES` if the delegate for the application *`sender`* handles the key specified by *`key`*, which means it can get or set the scriptable property or element that corresponds to that key. The application implements methods for each of the keys that it handles, where the method name matches the key.

For example, a scriptable application that doesn't use Cocoa's document-based application architecture can implement this method to supply its own document ordering. Such an application might want to do this because the standard application delegate expects to work with a document-based application. The TextEdit application (whose source is distributed with Mac OS X developer tools) provides the following implementation:

```
return [key isEqualToString:@"orderedDocuments"];
```

TextEdit then implements the `orderedDocuments` method in its controller class to return an ordered list of documents. An application with its own window ordering might add a test for the key `orderedWindows` so that its delegate can provide its own version of `orderedWindows`.

Important: Cocoa scripting does not invoke this method for script commands other than `get` or `set`. For information on working with other commands, see Script Commands in *Cocoa Scripting Guide*.

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

See Also
- `orderedDocuments` (page 38)
- `orderedWindows` (page 39)

Declared In
`NSApplicationScripting.h`

application:openFile:

Tells the delegate to open a single file.

```
- (BOOL)application:(NSApplication *)theApplication openFile:(NSString *)filename
```

Parameters
`theApplication`
The application object associated with the delegate.

`filename`
The name of the file to open.

Return Value
`YES` if the file was successfully opened or `NO` if it was not.

Discussion
Sent directly by *`theApplication`* to the delegate. The method should open the file *`filename`*, returning `YES` if the file is successfully opened, and `NO` otherwise. If the user started up the application by double-clicking a file, the delegate receives the `application:openFile:` message before receiving `applicationDidFinishLaunching:` (page 69). (`applicationWillFinishLaunching:` (page 75) is sent before `application:openFile:`.)

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

See Also
- `application:openFileWithoutUI:` (page 64)
- `application:openTempFile:` (page 65)
- `applicationOpenUntitledFile:` (page 71)

Declared In
`NSApplication.h`

application:openFiles:

Tells the delegate to open multiple files.

```
- (void)application:(NSApplication *)sender openFiles:(NSArray *)filenames
```

Parameters

sender
The application object associated with the delegate.

filenames
An array of `NSString` objects containing the names of the files to open..

Discussion
Identical to `application:openFile:` (page 63) except that the receiver opens multiple files corresponding to the file names in the *filenames* array. Delegates should invoke the `replyToOpenOrPrint:` (page 44) method upon success or failure, or when the user cancels the operation.

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

Declared In
`NSApplication.h`

application:openFileWithoutUI:

Tells the delegate to open a file programmatically.

```
- (BOOL)application:(id)sender openFileWithoutUI:(NSString *)filename
```

Parameters

sender
The object that sent the command.

filename
The name of the file to open.

Return Value
`YES` if the file was successfully opened or `NO` if it was not.

Discussion
Sent directly by *sender* to the delegate to request that the file *filename* be opened as a linked file. The method should open the file without bringing up its application's user interface—that is, work with the file is under programmatic control of *sender*, rather than under keyboard control of the user.

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

See Also
- `application:openFile:` (page 63)
- `application:openTempFile:` (page 65)
- `applicationOpenUntitledFile:` (page 71)
- `application:printFile:` (page 66)

Declared In
`NSApplication.h`

application:openTempFile:

Tells the delegate to open a temporary file.

```
- (BOOL)application:(NSApplication *)theApplication openTempFile:(NSString *)filename
```

Parameters

theApplication
The application object associated with the delegate.

filename
The name of the temporary file to open.

Return Value
`YES` if the file was successfully opened or `NO` if it was not.

Discussion
Sent directly by *theApplication* to the delegate. The method should attempt to open the file *filename*, returning `YES` if the file is successfully opened, and `NO` otherwise.

By design, a file opened through this method is assumed to be temporary—it's the application's responsibility to remove the file at the appropriate time.

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

See Also
- `application:openFile:` (page 63)
- `application:openFileWithoutUI:` (page 64)
- `applicationOpenUntitledFile:` (page 71)

Declared In
`NSApplication.h`

application:printFile:

Sent when the user starts up the application on the command line with the `-NSPrint` option.

```
- (BOOL)application:(NSApplication *)theApplication printFile:(NSString *)filename
```

Parameters

theApplication

The application object that is handling the printing.

filename

The name of the file to print.

Return Value

`YES` if the file was successfully printed or `NO` if it was not.

Discussion

This message is sent directly by *theApplication* to the delegate. The application terminates (using the `terminate:` (page 57) method) after this method returns.

If at all possible, this method should print the file without displaying the user interface. For example, if you pass the `-NSPrint` option to the TextEdit application, TextEdit assumes you want to print the entire contents of the specified file. However, if the application opens more complex documents, you may want to display a panel that lets the user choose exactly what they want to print.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- application:openFileWithoutUI:` (page 64)

Declared In

`NSApplication.h`

application:printFiles:withSettings:showPrintPanels:

Prints a group of files.

```
- (NSApplicationPrintReply)application:(NSApplication *)application
    printFiles:(NSArray *)fileNames withSettings:(NSDictionary *)printSettings
    showPrintPanels:(BOOL)showPrintPanels
```

Parameters

application

The application object that is handling the printing.

fileNames

An array of `NSString` objects, each of which contains the name of a file to print.

printSettings

Para

showPrintPanels

Para

Return Value
A constant indicating whether printing was successful. For a list of possible values, see "Constants" (page 77).

Discussion
Sent to the delegate by *application*. The method should print the files named in the *fileNames* array using *printSettings*, a dictionary containing `NSPrintInfo`-compatible print job attributes. The *showPrintPanels* argument is a flag indicating whether or not a print panel should be presented for each file being printed. If it is NO, no print panels should be presented (but print progress indicators should still be presented).

Return `NSPrintingReplyLater` if the result of printing cannot be returned immediately, for example, if printing will cause the presentation of a sheet. If your method returns `NSPrintingReplyLater` it must always invoke the `NSApplication` method `replyToOpenOrPrint:` when the entire print operation has been completed, successfully or not.

This delegate method replaces `application:printFiles:` (page 89), which is now deprecated. If your application delegate only implements the deprecated method, it is still invoked, and `NSApplication` uses private functionality to arrange for the print settings to take effect.

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

Declared In
`NSApplication.h`

application:willPresentError:

Sent to the delegate before the specified application presents an error message to the user.

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

Parameters

application
: The application object associated with the delegate.

error
: The error object that is used to construct the error message. Your implementation of this method can return a new `NSError` object or the same one in this parameter.

Return Value
The error object to display.

Discussion
You can implement this delegate method to customize the presentation of any error presented by your application, as long as no code in your application overrides either of the `NSResponder` methods `presentError:modalForWindow:delegate:didPresentSelector:contextInfo:` or `presentError:` in a way that prevents errors from being passed down the responder chain to the application object.

Your implementation of this delegate method can examine *error* and, if its localized description or recovery information is unhelpfully generic, return an error object with specific localized text that is more suitable for presentation in alert sheets and dialogs. If you do this, always use the domain and error code of the `NSError` object to distinguish between errors whose presentation you want to customize and those you do not. Don't

make decisions based on the localized description, recovery suggestion, or recovery options because parsing localized text is problematic. If you decide not to customize the error presentation, just return the passed-in error object.

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

Declared In
NSApplication.h

applicationDidBecomeActive:

Sent by the default notification center immediately after the application becomes active.

```
- (void)applicationDidBecomeActive:(NSNotification *)aNotification
```

Parameters

aNotification

A notification of the type `NSApplicationDidBecomeActiveNotification` (page 84). Calling the `object` method of this notification returns the `NSApplication` object itself.

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

See Also
- `applicationDidFinishLaunching:` (page 69)
- `applicationDidResignActive:` (page 70)
- `applicationWillBecomeActive:` (page 74)

Declared In
NSApplication.h

applicationDidChangeScreenParameters:

Sent by the default notification center when the configuration of the displays attached to the computer is changed (either programmatically or when the user changes settings in the Displays control panel).

```
- (void)applicationDidChangeScreenParameters:(NSNotification *)aNotification
```

Parameters

aNotification

A notification of the type `NSApplicationDidChangeScreenParametersNotification` (page 84). Calling the `object` method of this notification returns the `NSApplication` object itself.

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

Declared In
NSApplication.h

applicationDidFinishLaunching:

Sent by the default notification center after the application has been launched and initialized but before it has received its first event.

```
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
```

Parameters

aNotification

A notification of the type `NSApplicationDidFinishLaunchingNotification` (page 85). Calling the `object` method of this notification returns the `NSApplication` object itself.

Discussion

Delegates can implement this method to perform further initialization. This method is called after the application's main run loop has been started but before it has processed any events. If the application was launched by the user opening a file, the delegate's `application:openFile:` method is called before this method. If you want to perform initialization before any files are opened, implement the `applicationWillFinishLaunching:` (page 75) method in your delegate, which is called before `application:openFile:`.)

Availability

Available in Mac OS X v10.0 and later.

See Also

- `finishLaunching` (page 32)
- `applicationWillFinishLaunching:` (page 75)
- `applicationDidBecomeActive:` (page 68)
- `application:openFile:` (page 63)

Declared In

`NSApplication.h`

applicationDidHide:

Sent by the default notification center immediately after the application is hidden.

```
- (void)applicationDidHide:(NSNotification *)aNotification
```

Parameters

aNotification

A notification of the type `NSApplicationDidHideNotification` (page 85). Calling the `object` method of this notification returns the `NSApplication` object itself.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `applicationWillHide:` (page 75)
- `applicationDidUnhide:` (page 70)
- `hide:` (page 33)

Declared In

`NSApplication.h`

applicationDidResignActive:

Sent by the default notification center immediately after the application is deactivated.

```
- (void)applicationDidResignActive:(NSNotification *)aNotification
```

Parameters

aNotification

A notification of the type `NSApplicationDidResignActiveNotification` (page 85). Calling the `object` method of this notification returns the `NSApplication` object itself.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `applicationDidBecomeActive:` (page 68)
- `applicationWillResignActive:` (page 76)

Declared In

`NSApplication.h`

applicationDidUnhide:

Sent by the default notification center immediately after the application is made visible.

```
- (void)applicationDidUnhide:(NSNotification *)aNotification
```

Parameters

aNotification

A notification of the type `NSApplicationDidUnhideNotification` (page 85). Calling the `object` method of this notification returns the `NSApplication` object itself.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `applicationDidHide:` (page 69)
- `applicationWillUnhide:` (page 76)
- `unhide:` (page 58)

Declared In

`NSApplication.h`

applicationDidUpdate:

Sent by the default notification center immediately after the application object updates its windows.

```
- (void)applicationDidUpdate:(NSNotification *)aNotification
```

Parameters

aNotification

A notification of the type `NSApplicationDidUpdateNotification` (page 85). Calling the `object` method of this notification returns the `NSApplication` object itself.

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

See Also
- `applicationWillUpdate:` (page 77)
- `updateWindows` (page 59)

Declared In
`NSApplication.h`

applicationDockMenu:

Allows the delegate to supply a dock menu for the application dynamically.

```
- (NSMenu *)applicationDockMenu:(NSApplication *)sender
```

Parameters

sender
The application object associated with the delegate.

Return Value
The menu to display in the dock.

Discussion
You can also connect a menu in Interface Builder to the `dockMenu` outlet. A third way for your application to specify a dock menu is to provide an `NSMenu` in a nib.

If this method returns a menu, this menu takes precedence over the `dockMenu` in the nib.

The target and action for each menu item are passed to the dock. On selection of the menu item the dock messages your application, which should invoke `[NSApp sendAction:selector to:target from:nil]`.

To specify an `NSMenu` in a nib, you add the nib name to the `info.plist`, using the key `AppleDockMenu`. The nib name is specified without an extension. You then create a connection from the file's owner object (which by default is `NSApplication`) to the menu. Connect the menu to the `dockMenu` outlet of `NSApplication`. The menu is in its own nib file so it can be loaded lazily when the `dockMenu` is requested, rather than at launch time.

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

Declared In
`NSApplication.h`

applicationOpenUntitledFile:

Tells the delegate to open an untitled file.

```
- (BOOL)applicationOpenUntitledFile:(NSApplication *)theApplication
```

Parameters

theApplication
The application object associated with the delegate.

Return Value
YES if the file was successfully opened or NO if it was not.

Discussion
Sent directly by *theApplication* to the delegate to request that a new, untitled file be opened.

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

See Also
- application:openFile: (page 63)
- application:openFileWithoutUI: (page 64)
- application:openTempFile: (page 65)

Declared In
NSApplication.h

applicationShouldHandleReopen:hasVisibleWindows:

Sent by the application to the delegate prior to default behavior to reopen (rapp) AppleEvents.

```
- (BOOL)applicationShouldHandleReopen:(NSApplication *)theApplication
    hasVisibleWindows:(BOOL)flag
```

Parameters

theApplication
The application object.

flag
Indicates whether the NSApplication object found any visible windows in your application. You can use this value as an indication of whether the application would do anything if you return YES.

Return Value
YES if you want the application to perform its normal tasks or NO if you want the application to do nothing.

Discussion
These events are sent whenever the Finder reactivates an already running application because someone double-clicked it again or used the dock to activate it. By default the Application Kit will handle this event by checking whether there are any visible NSWindow (not NSPanel) objects, and, if there are none, it goes through the standard untitled document creation (the same as it does if *theApplication* is launched without any document to open). For most document-based applications, an untitled document will be created. The application delegate will also get a chance to respond to the normal untitled document delegate methods. If you implement this method in your application delegate, it will be called before any of the default behavior happens. If you return YES, then NSApplication will go on to do its normal thing. If you return NO, then NSApplication will do nothing. So, you can either implement this method, do nothing, and return NO if you do not want anything to happen at all (not recommended), or you can implement this method, handle the event yourself in some custom way, and return NO.

Note that what happens to minimized windows is not determined yet, but the intent is that *flag* being NO indicates whether the Application Kit will create a new window to satisfy the reopen event.

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

Declared In
NSApplication.h

applicationShouldOpenUntitledFile:

Invoked immediately before opening an untitled file.

```
- (BOOL)applicationShouldOpenUntitledFile:(NSApplication *)sender
```

Parameters

sender

The application object associated with the delegate.

Return Value
YES if the application should open a new untitled file or NO if it should not.

Discussion
Use this method to decide whether the application should open a new, untitled file. Note that applicationOpenUntitledFile: (page 71) is invoked if this method returns YES.

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

Declared In
NSApplication.h

applicationShouldTerminate:

Sent to notify the delegate that the application is about to terminate.

```
- (NSApplicationTerminateReply)applicationShouldTerminate:(NSApplication *)sender
```

Parameters

sender

The application object that is about to be terminated.

Return Value
One of the values defined in NSApplicationTerminateReply (page 80) constants indicating whether the application should terminate. For compatibility reasons, a return value of NO is equivalent to NSTerminateCancel, and a return value of YES is equivalent to NSTerminateNow.

Discussion
This method is typically called after the application's Quit or Exit command has been selected, or after the FOO method has been called. Generally, you should return NSTerminateNow to allow the termination to complete, but you can cancel the termination process or delay it somewhat as needed. For example, you might delay termination to finish processing some critical data but then terminate the application as soon as you are done by calling the replyToApplicationShouldTerminate: method.

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

See Also
- terminate: (page 57)
- applicationShouldTerminateAfterLastWindowClosed: (page 74)

- applicationWillTerminate: (page 76)

Declared In
NSApplication.h

applicationShouldTerminateAfterLastWindowClosed:

Invoked when the user closes the last window the application has open.

```
- (BOOL)applicationShouldTerminateAfterLastWindowClosed:(NSApplication
    *)theApplication
```

Parameters

theApplication

The application object whose last window was closed.

Return Value
NO if the application should not be terminated when its last window is closed; otherwise, YES to terminate the application.

Discussion
The application sends this message to your delegate when the application's last window is closed. It sends this message regardless of whether there are still panels open. (A panel in this case is defined as being an instance of NSPanel or one of its subclasses.)

If your implementation returns NO, control returns to the main event loop and the application is not terminated. If you return YES, your delegate's applicationShouldTerminate: method is subsequently invoked to confirm that the application should be terminated.

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

See Also
- terminate: (page 57)
- applicationShouldTerminate: (page 73)

Declared In
NSApplication.h

applicationWillBecomeActive:

Sent by the default notification center immediately before the application becomes active.

```
- (void)applicationWillBecomeActive:(NSNotification *)aNotification
```

Parameters

aNotification

A notification of the type NSApplicationWillBecomeActiveNotification (page 86). Calling the object method of this notification returns the NSApplication object itself.

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

See Also
- `applicationDidBecomeActive:` (page 68)
- `applicationWillFinishLaunching:` (page 75)
- `applicationWillResignActive:` (page 76)

Declared In
`NSApplication.h`

applicationWillFinishLaunching:

Sent by the default notification center immediately before the application object is initialized.

```
- (void)applicationWillFinishLaunching:(NSNotification *)aNotification
```

Parameters

aNotification

A notification of the type `NSApplicationWillFinishLaunchingNotification` (page 86). Calling the `object` method of this notification returns the `NSApplication` object itself.

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

See Also
- `applicationDidFinishLaunching:` (page 69)
- `applicationWillBecomeActive:` (page 74)
- `finishLaunching` (page 32)

Declared In
`NSApplication.h`

applicationWillHide:

Sent by the default notification center immediately before the application is hidden.

```
- (void)applicationWillHide:(NSNotification *)aNotification
```

Parameters

aNotification

A notification of the type `NSApplicationWillHideNotification` (page 86). Calling the `object` method of this notification returns the `NSApplication` object itself.

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

See Also
- `applicationDidHide:` (page 69)
- `hide:` (page 33)

Declared In
`NSApplication.h`

applicationWillResignActive:

Sent by the default notification center immediately before the application is deactivated.

```
- (void)applicationWillResignActive:(NSNotification *)aNotification
```

Parameters

aNotification

A notification of the type `NSApplicationWillResignActiveNotification` (page 86). Calling the `object` method of this notification returns the `NSApplication` object itself.

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

See Also
- `applicationWillBecomeActive:` (page 74)
- `applicationDidResignActive:` (page 70)

Declared In
`NSApplication.h`

applicationWillTerminate:

Sent by the default notification center immediately before the application terminates.

```
- (void)applicationWillTerminate:(NSNotification *)aNotification
```

Parameters

aNotification

A notification of the type `NSApplicationWillTerminateNotification` (page 87). Calling the `object` method of this notification returns the `NSApplication` object itself.

Discussion
Your delegate can use this method to perform any final cleanup before the application terminates.

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

See Also
- `applicationShouldTerminate:` (page 73)
- `terminate:` (page 57)

Declared In
`NSApplication.h`

applicationWillUnhide:

Sent by the default notification center immediately after the application is unhidden.

```
- (void)applicationWillUnhide:(NSNotification *)aNotification
```

Parameters

aNotification

A notification of the type `NSApplicationWillUnhideNotification` (page 87). Calling the `object` method of this notification returns the `NSApplication` object itself.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `unhide:` (page 58)
- `applicationDidUnhide:` (page 70)
- `applicationWillHide:` (page 75)

Declared In

`NSApplication.h`

applicationWillUpdate:

Sent by the default notification center immediately before the application object updates its windows.

```
- (void)applicationWillUpdate:(NSNotification *)aNotification
```

Parameters

aNotification

A notification of the type `NSApplicationWillUpdateNotification` (page 87). Calling the `object` method of this notification returns the `NSApplication` object itself.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `applicationDidUpdate:` (page 70)
- `updateWindows` (page 59)

Declared In

`NSApplication.h`

Constants

Return values for modal operations

These are possible return values for `runModalForWindow:` (page 46) and `runModalSession:` (page 47).

```
enum {
    NSRunStoppedResponse    = (-1000),
    NSRunAbortedResponse    = (-1001),
    NSRunContinuesResponse  = (-1002)
};
```

Constants

`NSRunStoppedResponse`

Modal session was broken with `stopModal` (page 55).

Available in Mac OS X v10.0 and later.

Declared in `NSApplication.h`.

`NSRunAbortedResponse`

Modal session was broken with `abortModal` (page 22).

Available in Mac OS X v10.0 and later.

Declared in `NSApplication.h`.

`NSRunContinuesResponse`

Modal session is continuing (returned by `runModalSession:` (page 47) only).

Available in Mac OS X v10.0 and later.

Declared in `NSApplication.h`.

Discussion

The system also reserves all values below these.

Declared In

`NSApplication.h`

NSUpdateWindowsRunLoopOrdering

This constant is used by the `NSRunLoop` method `performSelector:target:argument:order:modes:`.

```
enum {
    NSUpdateWindowsRunLoopOrdering = 500000
};
```

Constants

`NSUpdateWindowsRunLoopOrdering`

Run-loop message priority for handling window updates.

Available in Mac OS X v10.0 and later.

Declared in `NSApplication.h`.

Declared In

`NSApplication.h`

NSApp

A global constant for the shared application instance.

```
id NSApp
```

Constants

`NSApp`

Global constant for the shared application instance.

Available in Mac OS X v10.0 and later.

Declared in `NSApplication.h`.

Discussion

This variable designates the shared application object, created by the sharedApplication (page 21) method.

Declared In

`NSApplication.h`

NSRequestUserAttentionType

These constants specify the level of severity of a user attention request and are used by `cancelUserAttentionRequest:` (page 27) and `requestUserAttention:` (page 45).

```
typedef enum {
    NSCriticalRequest = 0,
    NSInformationalRequest = 10
} NSRequestUserAttentionType;
```

Constants

`NSCriticalRequest`

The user attention request is a critical request.

The dock icon will bounce until either the application becomes active or the request is canceled.

Available in Mac OS X v10.1 and later.

Declared in `NSApplication.h`.

`NSInformationalRequest`

The user attention request is an informational request.

The dock icon will bounce for one second. The request, though, remains active until either the application becomes active or the request is canceled.

Available in Mac OS X v10.1 and later.

Declared in `NSApplication.h`.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`NSApplication.h`

NSApplicationDelegateReply

These constants indicate whether or not a copy or print operation was successful, was cancelled, or failed. These constants are used by the `replyToOpenOrPrint:` (page 44) method.

```
typedef enum NSApplicationDelegateReply {
    NSApplicationDelegateReplySuccess = 0,
    NSApplicationDelegateReplyCancel = 1,
    NSApplicationDelegateReplyFailure = 2
} NSApplicationDelegateReply;
```

Constants

`NSApplicationDelegateReplySuccess`

Indicates the operation succeeded.

Available in Mac OS X v10.3 and later.

Declared in `NSApplication.h`.

`NSApplicationDelegateReplyCancel`

Indicates the user cancelled the operation.

Available in Mac OS X v10.3 and later.

Declared in `NSApplication.h`.

`NSApplicationDelegateReplyFailure`

Indicates an error occurred processing the operation.

Available in Mac OS X v10.3 and later.

Declared in `NSApplication.h`.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`NSApplication.h`

NSApplicationTerminateReply

These constants define whether an application should terminate and are used by `applicationShouldTerminate:` (page 73).

```
typedef enum NSApplicationTerminateReply {
    NSTerminateCancel = 0,
    NSTerminateNow    = 1,
    NSTerminateLater  = 2
} NSApplicationTerminateReply;
```

Constants

`NSTerminateNow`

It is OK to proceed with termination.

Available in Mac OS X v10.0 and later.

Declared in `NSApplication.h`.

`NSTerminateCancel`

The application should not be terminated.

Available in Mac OS X v10.0 and later.

Declared in `NSApplication.h`.

`NSTerminateLater`
: It may be OK to proceed with termination later. Returning this value causes Cocoa to run the run loop in the `NSModalPanelRunLoopMode` until your application subsequently calls `replyToApplicationShouldTerminate:` (page 43) with the value `YES` or `NO`. This return value is for delegates that need to provide document modal alerts (sheets) in order to decide whether to quit.

  Available in Mac OS X v10.0 and later.

  Declared in `NSApplication.h`.

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

Declared In
`NSApplication.h`

NSApplicationPrintReply

These constants are returned by `application:printFiles:withSettings:showPrintPanels:` (page 66).

```
typedef enum NSApplicationPrintReply {
    NSPrintingCancelled  = 0,
    NSPrintingSuccess    = 1,
    NSPrintingFailure    = 3,
    NSPrintingReplyLater = 2
} NSApplicationPrintReply;
```

Constants

`NSPrintingCancelled`
: Printing was cancelled.

  Available in Mac OS X v10.4 and later.

  Declared in `NSApplication.h`.

`NSPrintingSuccess`
: Printing was successful.

  Available in Mac OS X v10.4 and later.

  Declared in `NSApplication.h`.

`NSPrintingFailure`
: Printing failed.

  Available in Mac OS X v10.4 and later.

  Declared in `NSApplication.h`.

`NSPrintingReplyLater`
: The result of printing cannot be returned immediately, for example, if printing will cause the presentation of a sheet. If your method returns `NSPrintingReplyLater` it must always invoke `replyToOpenOrPrint:` (page 44) when the entire print operation has been completed, successfully or not.

  Declared in `NSApplication.h`.

  Available in Mac OS X v10.4 and later.

Declared In
`NSApplication.h`

Run loop modes

These loop mode constants are defined by `NSApplication`.

```
NSString *NSModalPanelRunLoopMode;
NSString *NSEventTrackingRunLoopMode;
```

Constants

`NSEventTrackingRunLoopMode`

A run loop should be set to this mode when tracking events modally, such as a mouse-dragging loop.

Available in Mac OS X v10.0 and later.

Declared in `NSApplication.h`.

`NSModalPanelRunLoopMode`

A run loop should be set to this mode when waiting for input from a modal panel, such as `NSSavePanel` or `NSOpenPanel`.

Available in Mac OS X v10.0 and later.

Declared in `NSApplication.h`.

Declared In

`NSApplication.h`

NSAppKitVersionNumber

This constant identifies the installed version of the Application Kit framework.

```
const double NSAppKitVersionNumber;
```

Constants

`NSAppKitVersionNumber`

This value corresponds to one of the constants defined in "Application Kit framework version numbers" (page 82).

Available in Mac OS X v10.1 and later.

Declared in `NSApplication.h`.

Declared In

`NSApplication.h`

Application Kit framework version numbers

You can use the following constants to determine if you are using a version of the Application Kit framework newer than the version delivered in Mac OS X v10.0.

```
#define NSAppKitVersionNumber10_0 577
#define NSAppKitVersionNumber10_1 620
#define NSAppKitVersionNumber10_2 663
#define NSAppKitVersionNumber10_2_3 663.6
#define NSAppKitVersionNumber10_3 743
#define NSAppKitVersionNumber10_3_2 743.14
#define NSAppKitVersionNumber10_3_3 743.2
#define NSAppKitVersionNumber10_3_5 743.24
#define NSAppKitVersionNumber10_3_7 743.33
#define NSAppKitVersionNumber10_3_9 743.36
#define NSAppKitVersionNumber10_4 824
```

Constants

`NSAppKitVersionNumber10_0`

The Application Kit framework included in Mac OS X v10.0.

Available in Mac OS X v10.1 and later.

Declared in `NSApplication.h`.

`NSAppKitVersionNumber10_1`

The Application Kit framework included in Mac OS X v10.1.

Available in Mac OS X v10.2 and later.

Declared in `NSApplication.h`.

`NSAppKitVersionNumber10_2`

The Application Kit framework included in Mac OS X v10.2.

Available in Mac OS X v10.3 and later.

Declared in `NSApplication.h`.

`NSAppKitVersionNumber10_2_3`

The Application Kit framework included in Mac OS X v10.2.3.

Available in Mac OS X v10.3 and later.

Declared in `NSApplication.h`.

`NSAppKitVersionNumber10_3`

The Application Kit framework included in Mac OS X v10.3.

Available in Mac OS X v10.4 and later.

Declared in `NSApplication.h`.

`NSAppKitVersionNumber10_3_2`

The Application Kit framework included in Mac OS X v10.3.2.

Available in Mac OS X v10.4 and later.

Declared in `NSApplication.h`.

`NSAppKitVersionNumber10_3_3`

The Application Kit framework included in Mac OS X v10.3.3.

Available in Mac OS X v10.4 and later.

Declared in `NSApplication.h`.

`NSAppKitVersionNumber10_3_5`

The Application Kit framework included in Mac OS X v10.3.5.

Available in Mac OS X v10.4 and later.

Declared in `NSApplication.h`.

`NSAppKitVersionNumber10_3_7`
: The Application Kit framework included in Mac OS X v10.3.7.

  Available in Mac OS X v10.5 and later.

  Declared in `NSApplication.h`.

`NSAppKitVersionNumber10_3_9`
: The Application Kit framework included in Mac OS X v10.3.9.

  Available in Mac OS X v10.5 and later.

  Declared in `NSApplication.h`.

`NSAppKitVersionNumber10_4`
: The Application Kit framework included in Mac OS X v10.4.

  Available in Mac OS X v10.5 and later.

  Declared in `NSApplication.h`.

Declared In
`NSApplication.h`

Notifications

These notifications apply to `NSApplication`. See "Notifications" in `NSWorkspace` for additional, similar notifications.

NSApplicationDidBecomeActiveNotification

Posted immediately after the application becomes active.

The notification object is `NSApp`. This notification does not contain a *`userInfo`* dictionary.

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

Declared In
`NSApplication.h`

NSApplicationDidChangeScreenParametersNotification

Posted when the configuration of the displays attached to the computer is changed.

The configuration change can be made either programmatically or when the user changes settings in the Displays control panel. The notification object is `NSApp`. This notification does not contain a *`userInfo`* dictionary.

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

Declared In
`NSApplication.h`

NSApplicationDidFinishLaunchingNotification

Posted at the end of the `finishLaunching` (page 32) method to indicate that the application has completed launching and is ready to run.

The notification object is `NSApp`. This notification does not contain a *`userInfo`* dictionary.

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

Declared In
`NSApplication.h`

NSApplicationDidHideNotification

Posted at the end of the `hide:` (page 33) method to indicate that the application is now hidden.

The notification object is `NSApp`. This notification does not contain a *`userInfo`* dictionary.

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

Declared In
`NSApplication.h`

NSApplicationDidResignActiveNotification

Posted immediately after the application gives up its active status to another application.

The notification object is `NSApp`. This notification does not contain a *`userInfo`* dictionary.

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

Declared In
`NSApplication.h`

NSApplicationDidUnhideNotification

Posted at the end of the `unhideWithoutActivation` (page 59) method to indicate that the application is now visible.

The notification object is `NSApp`. This notification does not contain a *`userInfo`* dictionary.

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

Declared In
`NSApplication.h`

NSApplicationDidUpdateNotification

Posted at the end of the `updateWindows` (page 59) method to indicate that the application has finished updating its windows.

The notification object is `NSApp`. This notification does not contain a *`userInfo`* dictionary.

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

Declared In
`NSApplication.h`

NSApplicationWillBecomeActiveNotification

Posted immediately after the application becomes active.

The notification object is `NSApp`. This notification does not contain a *`userInfo`* dictionary.

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

Declared In
`NSApplication.h`

NSApplicationWillFinishLaunchingNotification

Posted at the start of the `finishLaunching` (page 32) method to indicate that the application has completed its initialization process and is about to finish launching.

The notification object is `NSApp`. This notification does not contain a *`userInfo`* dictionary.

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

Declared In
`NSApplication.h`

NSApplicationWillHideNotification

Posted at the start of the `hide:` (page 33) method to indicate that the application is about to be hidden.

The notification object is `NSApp`. This notification does not contain a *`userInfo`* dictionary.

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

Declared In
`NSApplication.h`

NSApplicationWillResignActiveNotification

Posted immediately before the application gives up its active status to another application.

The notification object is `NSApp`. This notification does not contain a *`userInfo`* dictionary.

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

Declared In
NSApplication.h

NSApplicationWillTerminateNotification

Posted by the terminate: (page 57) method to indicate that the application will terminate.

Posted only if the delegate method applicationShouldTerminate: (page 73) returns YES. The notification object is NSApp. This notification does not contain a *userInfo* dictionary.

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

Declared In
NSApplication.h

NSApplicationWillUnhideNotification

Posted at the start of the unhideWithoutActivation (page 59) method to indicate that the application is about to become visible.

The notification object is NSApp. This notification does not contain a *userInfo* dictionary.

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

Declared In
NSApplication.h

NSApplicationWillUpdateNotification

Posted at the start of the updateWindows (page 59) method to indicate that the application is about to update its windows.

The notification object is NSApp. This notification does not contain a *userInfo* dictionary.

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

Declared In
NSApplication.h

Deprecated NSApplication Methods

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

Deprecated in Mac OS X v10.4

application:printFiles:

(Deprecated in Mac OS X v10.4. Use `application:printFiles:withSettings:showPrintPanels:` (page 66) instead.)

```
- (void)application:(NSApplication *)sender printFiles:(NSArray *)filenames
```

Discussion
Identical to `application:printFile:` (page 66) except that the receiver prints multiple files corresponding to the file names in the *filenames* array.

Delegates should invoke the `replyToOpenOrPrint:` (page 44) method upon success or failure, or when the user cancels the operation.

Availability
Deprecated in Mac OS X v10.4.

Declared In
`NSApplication.h`

Document Revision History

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

Date	Notes
2009-02-04	Added dockTile method description.
2008-10-15	Updated the list of AppKit version constants. Clarified information surrounding the use of modal panels.
2007-04-24	Made editorial improvements.
	Added `NSTerminateCancel` (page 80) to discussion of `terminate:` (page 57) method.
2006-05-23	Corrected typo.
	First publication of this content as a separate document.
	Revised description for application:delegateHandlesKey: (page 62) to indicate it is invoked by Cocoa scripting support only in handling the `get` and `set` script commands.

Document Revision History

Index

A

B

C

D

E

F

H

I

K

M

N

O

P

R

S

T

U

V

W